A functional multiprocessor system for real-time digital signal processing.

Sulley, C. E.

Award date:
1985

Awarding institution:
University of Bath

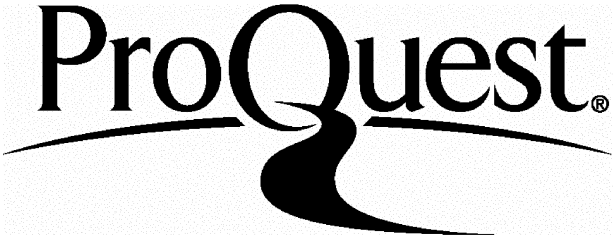
A FUNCTIONAL MULTIPROCESSOR SYSTEM FOR REAL-TIME DIGITAL SIGNAL PROCESSING

Submitted by: C E Sulley

for the degree of Ph D

of the University of Bath

1985



LIST OF CONTENTS

LIST OF CONTENTS CONTINUED

LIST OF FIGURES

LIST OF FIGURES CONTINUED

LIST OF FIGURES CONTINUED

LIST OF TABLES

ABSTRACT

This thesis is concerned primarily with the architecture of Digital Signal Computers. The work is supported by the design, development and application of a novel Digital Signal Computer system, the MAC68.

The MAC68 is a Functional Multiprocessor, using two independent processors, one of which executes general-purpose tasks, and the other executes sequences of arithmetic. The particular MAC68 design was arrived at after careful evaluation of existing Digital Signal Computer architectures. MAC68 features are fully evaluated via its application to the Sub-Band Coding of speech, and in particular by the development of a 16Kb/s Sub-band Coder using six sub-bands. MAC68 performance was found to be comparable to that of current DSP micros for basic digital filter tasks, and superior for FFT tasks.

The MAC68 architecture is a balance of high-speed arithmetic and general-purpose capabilities, and is likely to have a greater range of application than General-Purpose micros or DSP micros used alone. Suggestions are put forward for MAC68 enhancements utilising state-of-the-art hardware and software technologies.

Because of the current widespread use of General-Purpose micros, and because of the possible performance gains to be had with the MAC68-type architecture, it is thought that MAC68 architectural concepts will be of value in the design of future high-performance Digital Signal Computer systems.

ACKNOWLEDGEMENTS

I would like to thank Mr J D Martin of the University of Bath, and Dr N Kingsbury of the University of Cambridge (formerly of Marconi Secure Radio Systems Limited, Portsmouth), for their generous help and supervision. I would also like to thank other post-graduate students at the University of Bath and Dr I Jenkins of Marconi Secure Radio Systems Limited, Portsmouth, for useful discussions and support.

Special thanks to my family for their boundless encouragement, and in particular to my father for producing most of the Thesis figures and tables. Thanks also to Julie Bailey for the typing of this Thesis.

Finally, I would like to thank the Science and Engineering Research Council and Marconi Secure Radio Systems Limited, Portsmouth, for providing the financial support for this work.

ABBREVIATIONS

A/D	Analog-to-Digital Convertor.
APCM	Adaptive Pulse Code Modulation.
AU	Arithmetic Unit.
CISC	Complex Instruction Set Computer.
D/A	Digital-to-Analog Convertor.
DSC	Digital Signal Computer.
DSP	Digital Signal Processing.
FFT	Fast Fourier Transform.
FIR	Finite Impulse Response.
FM	Functional Multiprocessor.
GP	General-Purpose.
IIR	Infinite Impulse Response.
I/O	Input/Output.
ISR	Interrupt Service Routine.
MAC	Multiplier-Accumulator.
MM	Modular Multiprocessor.
QMF	Quadrature Mirror Filter.
RAM	Random Access Memory.
RISC	Reduced Instruction Set Computer.
S	Single-instruction Stream.
SBC	Sub-Band Coder.
S/H	Sample-Hold.
TQMF-BB	Tree Quadrature Mirror Filter Building Block.
VLSI	Very Large Scale Integration.

1. INTRODUCTION

This chapter introduces the subject area for this thesis, namely Digital Signal Computer Architectures. This is done by firstly discussing the nature of real-time Digital Signal Processing, and then leading on to Digital Signal Computers. Having established what Digital Signal Computing is, the remainder of this chapter outlines the motivation for this study of Digital Signal Computer Architectures, and then summarises the contents of the remaining chapters.

1.1 Real-Time Digital Signal Processing

Because of the large range of sources of digital signals, the variety of techniques associated with Digital Signal Processing is correspondingly large. For example:-

- Doppler Processing of radar signals.
- Digital Filter banks for speech signals.
- Digital Image Spectral Analysis.

The following is confined to one-dimensional Digital Signal Processing, but with the distinction that processing must be carried out in 'real-time'. 'Real-time' in this context refers to the situation where the available processing time is restricted, usually by the bandwidth of the sampled signal. The advantage of performing Digital Signal Processing in real-time is mainly that processing can be maintained continuously, without signal data being lost or large memories being required. For example as in radar, sonar or telecommunication systems.

A general form for a Real-time Digital Signal Processing system is presented in Figure 1.1, where an analog signal is sampled, digital signal processing carried out, and the digital result converted back to analog form. The total available computation time for each digital sample in this type of system, is determined by the bandwidth of the analog signal being sampled i.e. for a bandwidth of F Hz, a sampling rate of 2F samples/sec is required to avoid aliasing distortion. For example, for speech bandwidths a computation time of approximately 100usec. is available for each digital sample, and for video bandwidths a mere 0.1usec. is available.

This discussion immediately leads on to the form of Real-time Digital Signal Processors, the technology for which has only been available in the last two decades. Examples of Real-time Digital Signal Processors are:-

- Supercomputers.
- Single-chip micros.
- Dedicated Emitter Coupled Logic systems.
- Charge-Couple devices.
- Surface Acoustic Wave Devices.

The programmable type of Real-time Digital Signal Processor we shall call a Digital Signal Computer system, the remainder of this thesis is concerned only with this type of system.

1.2 Digital Signal Computer Systems

Figure 1.2 illustrates the form for a typical Digital Signal Computer system. Digital samples are represented internally by fixed point or floating point numbers. The single most important feature of Digital Signal Computer systems is that they can be reconfigured for different applications by a change of program only, this feature gives Digital Signal Computer systems an economic advantage over dedicated counterparts.

Consider now the idea of a Digital Signal Computer system architecture. This we shall describe as the interconnection and functionality of system building blocks, specifically for programmed Digital Signal Processing. Examples of system building blocks are:-

- Hardware multiplier.
- Address adder.
- Program memory.
- Complete computer.

In designing a Digital Signal Computer system architecture, Digital Signal Processing application algorithm must be studied in detail if the range of application of a system is to be maximised. The greater this range, the more cost-effective the design.

The attraction of using Digital Signal Computers has come about with advances in device technologies. Consider now the scale of these advances. True Digital Signal Computers originated in the early 1970s, a typical example being the FDP (G-3). The FDP was an 18-bit machine using Emitter Coupled Logic (ECL), and was capable of executing a fixed point multiplication in 450nsec, and a fixed point 1024-point complex Fast Fourier Transform (FFT) in 5.5msec. To achieve such performance 10,000 Integrated Circuit packages were required.

Since the early 1970s progress in device technologies has been dramatic, for example the number of Metal Oxide Semiconductor (MOS) devices in general-purpose micros has increased by a factor of 200 since they were first introduced in 1971.

1971 Intel 4004. 2300 devices.

1984 Hewlett Packard micro. 450,000 devices.

The current state of Digital Signal Computer technology, is that a complete 16-bit Digital Signal Computer can fit on a single chip. A typical example is the Texas Instruments TMS32010, which can execute a fixed point bi-quad filter section in under 2.5usec and a fixed point 32-point complex FFT in under 500usec.

As there is no indication that the pace of technology will slow down, then future Digital Signal Computer architectures will undoubtedly be more complicated. It is therefore imperative that the relationship between architectures and algorithms is better understood. It is towards this better understanding that this thesis is directed. This work is supported primarily through the design, construction and application of a novel Digital Signal Computer system, the MAC68.

Consider now a framework for classifying Digital Signal Computer architectures.

When only one instruction stream is ever active within a system, we shall call this system a Single-Instruction (SI) stream system. Conversely, for more than one active instruction stream with a system, we shall call such a system a Multiple-Instruction (MI) stream system. Considering MI stream systems only, the SI stream units making up a system may be functional or identical to each other. In the former case we shall call such MI stream systems, Functional Multiprocessors (FM), and in the latter case, Modular Multiprocessors (MM).

By calling SI stream systems, S type, we now have three fundamental types, S, FM and MM types. These types can be used to form a three-dimensional classification framework with which to classify existing and projected Digital Signal Computer system architectures (see Figure 1.3). This classification is unique to this thesis.

NOTE: This classification is not suitable for data flow type systems which do not use instruction streams. As data flow systems have yet to be used extensively for real-time DSP this is not a serious disadvantage.

1.3 Thesis Organisation

Chapter 2 initiates this study by firstly filling in the details of the S-type classification. This is then followed by a detailed assessment of the performance of single-chip micros when executing Digital Signal Processing algorithms.

Directly from the summary of Chapter 2, Chapter 3 moves into the realm of Multiple-Instruction stream systems i.e. the MAC68. The motivation for the design of the MAC68 system, a Functional Multi-processor, is firstly discussed. The remainder of Chapter 3 then fully dicusses all aspects of MAC68 architectural features and programming techniques.

Chapter 4 details the full exercising of all MAC68 features via its application to Sub-band Coding. Sub-band Coding makes use of speech signal redundancy to produce lower bit-rates than conventional speech coding methods like Pulse Code Modulation (PCM). The nature of the Sub-band Coding algorithms used was such that all MAC68 architectural features were used, thus verifying the design and allowing a full investigation of weaknesses.

Chapter 5 firstly establishes that the current version of the MAC68 system has met the performance requirements initially established in Chapter 3. The remainder of the chapter then builds on experience obtained in using the MAC68 system, as well as using new ideas on High-level Languages, to propose an enhanced MAC68 system which would combine Digital Signal Processing performance with ease of program development.

Finally, in Chapter 6 the strengthes of the MAC68 concept are firmly established, and the future potential of the MAC68 in the field of Real-time Digital Signal Processing is discussed.

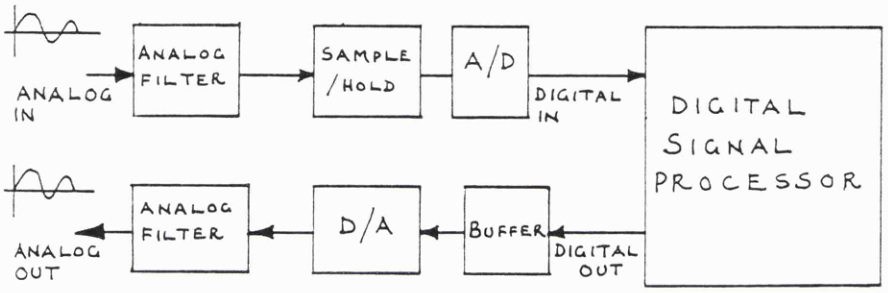


FIGURE 1.1. REAL-TIME DIGITAL SIGNAL PROCESSING SYSTEM.

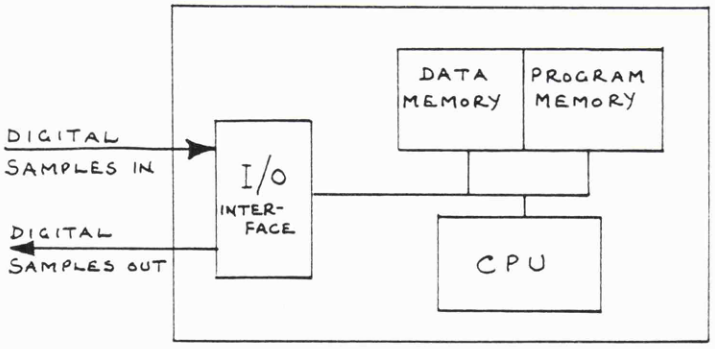
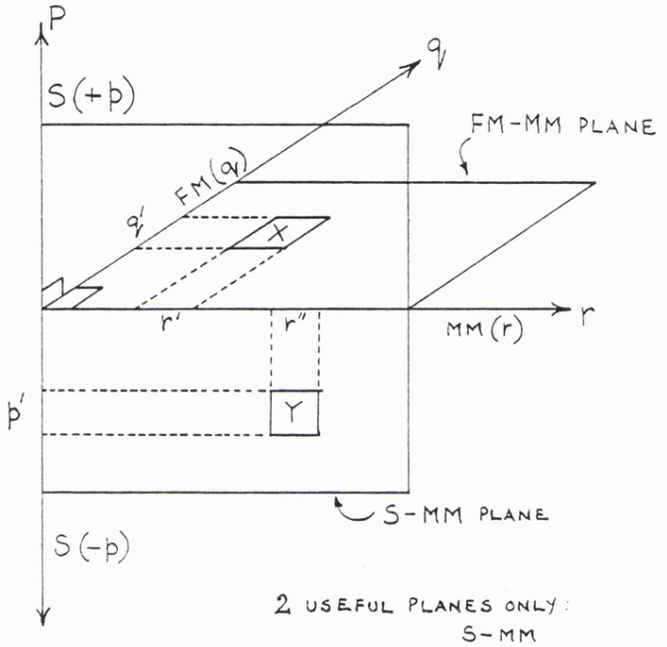


FIGURE 1.2. DIGITAL SIGNAL COMPUTER SYSTEM.



2 USEFUL PLANES ONLY:
S-MM
FM-MM

EXAMPLES

X: MULTIPLE FM (q') UNITS
MULTIPLE CONFIGURATION TYPE MM (r').

Y: MULTIPLE S (p') UNITS
MULTIPLE CONFIGURATION TYPE MM (r'').

FIGURE 1.3. THREE AXIS ARCHITECTURAL CLASSIFICATION OF DSC SYSTEMS

2. REVIEW OF SINGLE-INSTRUCTION STREAM DIGITAL SIGNAL COMPUTER SYSTEMS

This chapter is devoted entirely to looking at single-instruction stream Digital Signal Computer systems. Section 2.1 introduces a general classification scheme, which has come about as a result of analyses of the spectra of past, present and proposed single-instruction stream systems. In section 2.2 a detailed analysis of single-chip micros is carried out, making reference to the classification of 2.1. Finally in section 2.3, a general summary is produced, leading directly into chapter 3.

2.1 Classification of Single-Instruction Stream (S-Type) Digital Signal Computer Systems

The formation of a 'robust' classification of S-type systems, is complicated by the numerous interlocking influences. The following proposed classification is therefore informal, and is based on a 'nearest neighbour' approach, although attempting at the same time to illustrate the amount of parallelism used in system classes.

Table 2.1 illustrates the classification. Note that the system class using the least amount of parallelism is represented by S(0), and that generally as the magnitude of the index increases so too does the level of parallelism used. Also note that, the classification is almost circular in that S(+5) systems have much in common with S(-5) systems.

It should be pointed out that, the classification relates to architecture and not to performance, as levels of performance are affected by technological and algorithmic issues.

Each of the following sub-sections discusses an architectural feature of primary importance, such that when taken together they should demonstrate how the 'nearest neighbour' classification operates.

2.1.1 Instructions

Instruction formats and instruction execution sequences used by a system undoubtedly have the greatest effect on architectural typing. For this reason, and because two fundamental instruction types are observed in practice, RISC and CISC types (see below), the indexes of S-type systems are given positive and negative values, with positive indexes corresponding to CISC systems and negative indexes to RISC systems. Before elaborating further, consider some basic definitions.

2.1.1.1 Definitions

(a) For any S-type system three basic levels can be used [F-1]:

High Level Language =Language constructs

Image machine =Instructions, macroinstructions or machine instructions

Host machine =Microinstructions

The image machine is the lowest level of translation and the highest level of interpretation, and the mechanism is called the host machine. Consider then:-

(i) Image machine=Host machine, such that control is implemented with random logic (RL), and machine instructions are not interpreted. For example with microprocessors such as the Intel I8085 or Motorola MC6800.

(ii) Interpretation of image instructions using microcode sequences. Any number of interpretation levels can be used, but we shall only consider:-

- 1 level control scheme (M-1L)
- 2 level control scheme (M-2L)

These levels are the most commonly used.

Further we shall define microinstructions as being being [B-9,F-1]:-

(i) Horizontally orientated.
Where numerous micro-operations are activated by each microinstruction, as for example in the NEC7720.

(ii) Vertically orientated.
Where only a small number of micro-operations are activated by each microinstruction, for example in the Z80 microprocessor.

(b) Image Instruction complexity.

Two basic instruction set types are prevalent in current architectures, the Complex Instruction Set (CIS) and the Reduced Instruction Set (RIS) [B-2], although neither type has been precisely defined. Characteristics associated with these types are [B-2,P-1]:-

CIS instructions - Multiple cycle execution.
- Multiple sizes.
- Complex addressing modes.

RIS instructions - Most instructions execute in 1 cycle.
- Single instruction size.
- Access data memory with load and store instructions only.

2.1.1.2 RIS Computers (RISC) Versus CIS Computers (CISC)

The CISC approach is basically to support High Level Languages and Operating Systems via complex instructions, therefore effectively putting software in hardware. Some advantages of the CISC approach are:-

- Fewer fetches from instruction memory are required.
- Code density is higher, hence reducing memory requirements.

The alternative RISC approach is to use simple instructions (which can also be designed to support High Level Languages), and to rely on the availability of fast memory. Some advantages are:-

- Special-purpose hardware can be used to speed up the execution of particular functions. This hardware can be used instead of the hardware that would be needed for instruction control logic in a CISC system.

- Design is simplified.

Both methods also have disadvantages. RICSs being best suited to special-purpose applications which require only small amounts of memory. Whereas CISCs are more general-purpose and usually have facilities for handling large memory efficiently. Obvious disadvantages of the CISC method are:-

- Complex design.
- Increased execution times due to slow instruction cycles caused by complex instruction decode and control overheads.

Disadvantages of the RISC approach are:-

- The equivalent to CISC instructions may have to be implemented using subroutines, hence large register arrays (or large/fast memory) and efficient stacking may be needed as with the RISC1 [P-1].

- The RISC instruction cycle must be designed to accommodate the slowest executing function, such that faster functions must use a complete cycle (although this inefficiency can be reduced by letting slow functions take more than one cycle).

2.1.1.3 Complex Instruction Sets (CIS)

Architectures using Multi-cycle/multi-size instructions and with simple addressing modes, we shall call CISC-biased (CISC-b), for example most 8-bit micros such as the MC6800. Moving from CISC-b to CISC, instructions of greater complexity are used, consequently microcode sequences are used to ease implementation, although some CISC architectures do use random logic (RL) for instruction control, for example the Z8000 [T-1].

Considering only microcoded architectures, 2-level microcode schemes (M-2L) such as in the MC68000 are more spatially efficient than their 1-level counterparts [F-1], although 1-level schemes have shorter interpretation times. This is because there is a better mapping from image machine to host machine in 1-level schemes. Consider for example the 1-level scheme of the Hewlett Packard micro [G-1, G-2] which uses 302Kbits of microcode compared with the 22.5Kbits in the MC68000, but which has faster basic execution times.

The mapping from image machine to host machine can be improved further still, by using more parallelism via the use of horizontal microinstructions and pipelining. For example, basic cycle times can be reduced by pipelining activities in the Arithmetic Logic Unit (ALU). In extreme cases of pipelining, such as in supercomputers, asychronous pipelines are used so that image instructions can be executed out of sequence depending on the availability of host resources.

A further advantage of CISC systems, is that host machines can evolve with advances in technology while the image machine remains the same (or the instruction set is expanded), therefore software can be ported onto new host machines e.g. IBM 370 series [B-9].

2.1.1.4 Reduced Instruction Sets (RIS)

Architectures which use multi-cycle/single-size instructions we shall call RISC-biased (RISC-b), as control is relatively simple (the RISC-b example in Table 2.1 probably uses microcoding mainly for complete process scheduling and not for addressing). In moving to full RISC architectures, instruction control logic is at a minimum and RISC instructions are similar to CISC microinstructions.

As with CISC micromachines, the amount of parallelism in RISC architectures can be increased using horizontal microinstructions and pipelining. Unlike CISC architectures though, the use of parallelism in this way makes image machine programming more difficult. Consequently, a compromise between parallelism and programmability is often sought in RISC architectures, as for example in the IBM RSP [M-2]. Examples of where pipelining is used in RISC architectures is in single-chip Digital Signal Computers such as the BELL DSP and NEC-7720.

2.1.2 Memory Architecture and Instruction Pipelining

Consider two basic memory architectures:-

Von Neumann (VN) - instructions and data reside in the same memory space (therefore requiring a single memory interface).

Separate memory (SM) - instructions and data reside in separate areas, as used in special purpose architectures where the cost of two memory interfaces is acceptable. A good example of this is in Harvard machines [C-3].

All of the CISC architectures considered are of the VN type, as well as some of the RISC architectures. In all these cases it is desirable to overlap instruction fetches with instruction execution i.e. instruction pipelining. Clearly, for the VN case this is only possible if the memory interface logic and execution logic work independently, such that instructions are pre-fetched only when the execution logic does not require to access memory for data operands, as in the Intel 8086. When instruction pipelining is not used in a VN architecture, then the memory interface can become a bottleneck (a 'Von Neumann bottleneck' [B-7]), as for example in micros such as the MC6800.

Generally for CISC architectures a hierarchy of data memory is used to ensure that variables most frequently accessed are in faster memory, for example registers or data cache. The combination of this hierarchy with instruction pipelining minimises data and instruction movement overheads to the single memory space. For RISC-VN architectures, using simple addressing modes, there is a greater need for features such as instruction pipelining and large register areas. This is because the amount of data and instruction movement is likely to be greater than in CISC architectures executing equivalent instructions.

For RISC-SM architectures the above complexity does not arise, as instruction and data fetches can be fully overlapped without conflict. This instruction pipelining does cause branching problems though, for example for the IBM RSP user programs must be optimised in the vicinity of a branch instruction, and in the TMS32010 branching takes 2 cycles instead of the usual 1 cycle for instructions. Another feature of RISC-SM architectures is the greater freedom in choosing instruction formats i.e. instruction sizes do not have to conform to particular data sizes [C-3].

2.1.3 Multiple Arithmetic Units (AU)

CISC architectures employing multiple AUs are primarily for arithmetic intensive scientific computations, such as matrix multiplication or the solution of Partial Differential Equations, but can also be considered for Digital Signal Processing. The recently dismantled ILLIAC IV is a good

example (also called a Single-Instruction Multiple-Data (SIMD) machine), another example is the TI ASC which uses independent pipelines to each AU [R-1].

RISC architectures using multiple AUs are as equally specialised as their CISC counterparts. For example, the FDP uses four configurable AUs for operations such as complex arithmetic, digital filtering and multiple-precision arithmetic.

2.2 Digital Signal Processing (DSP) Performance of Single-Chip Micros

The objective of this section is to ascertain the DSP performance of single-chip micros, for the purpose of providing data and analyses for the design of a new Digital Signal Computer (DSC) system. Using the classification of section 2.1 this restricts the range to classes from S(-4) to S(+3), but because of insufficient data on classes S(+3), S(-1) and S(-2) only the following groups are considered:

(1) General-purpose micros of classes S(0), S(+1) and S(+2).
(2) DSP micros of classes S(-3) and S(-4).

Benchmarks used were for commonly used DSP tasks such as digital filtering and FFTs, and all benchmarks used fixed-point arithmetic. Also the following were not considered:-

(1) Input-Output Processing.
(2) Off-chip memory speeds i.e. Sufficiently fast memory was assumed to give minimum execution times.

(3) Variation in available clock speeds i.e. the most common clock speeds were chosen in each case.

2.2.1 General-Purpose (GP) Micros

Eleven commonly used GP micros were chosen, representing the spread in performance currently available. Using the classification of section 2.1, and introducing sub-classes also relating to complexity, we have:-

S(0): CISC-biased.

S(0)-I: I8085A, I8048, MC6800, Z80, I8051.

S(0)-II: MC6809, TMS9900.

S(1): CISC without microcoded control.

Z8000.

S(2): CISC with microcoded control.

I8086 [B-10], TMS9995, MC68000.

Intuitively, it would be thought that there is very little point in assessing the performance of 8 bit micros when the more powerful 16 bit micros are available. But, in architectural terms this is not the case, as the CISC philosophy is still unproven, and less complex 8 bit micros using faster clock speeds might prove superior for Digital Signal Processing tasks.

All GP micros selected are of the Von Neumann type, using the same memory area for data and programs. Table 2.2 summarises the major features of these GP micros, and also gives the on-chip multiplication capability (which is of greatest relevance to DSP, as will be seen later). On-chip 8-bit and 16-bit signed multiplication execution times were required for the analysis, consequently software routines were developed where these operations were not available as instructions. Full software multiplication routines developed assumed the use of minimum memory and maximum Hamming Weight [D-1].

2.2.1.1 Performance

Previously published DSP benchmarks for GP micros cover only the digital filtering capability of 16 bit micros [N-1]. This was considered insufficient, therefore the following benchmarks were written for the chosen GP micros (see Appendix I for further details):

BI: Eighth-order digital filter (16 bit operands).
This benchmark was derived from reference [N-1] and consists of four cascaded bi-quad sections (5 multiplications per section).

BII: 256-point complex Fast Fourier Transform (16 bit operands).
Constructed around the radix-2 FFT butterfly.

BIII: 32-word Cross-correlation (8 bit operands).

This benchmark consists essentially of sequences of sum-of-products calculations.

These benchmarks together represent a reasonable cross-section of DSP tasks, mixing:

- Real and complex arithmetic sequences.
- Complicated address calculations.
- Multiple-level loops.
- Fixed-point arithmetic using 16-bit and 8-bit operands.

From Table 2.2 it is clear that benchmark results could be made more useful if the use of some form of off-chip hardware multiplier were assumed i.e. bringing the multiplication time down to approximately the same as other basic operations. The TRW MPY-16 (50 nsec execution time) was chosen as a suitable hardware multiplier. A multiplier-accumulator such as the TRW TDC1010J could have been chosen instead, but the additional complexity of having to load arithmetic instructions would have blurred performance results.

It was assumed that the MPY-16 was sufficiently fast so that once two operands had been loaded and execution initiated, the result can be retrieved at any subsequent time. This is because the MPY-16 execution time is less than the basic instruction cycle time of any of the GP micros. Note that the MPY-16 is treated as locations in memory (see [G-4] for details).

Benchmark programs were written for the GP micros, with and without the assumption of the use of an off-chip multiplier, giving a total of 58 benchmark programs.

NOTE: Benchmarks BII and BIII were not written for the INTEL 8048 and INTEL 8051.

In view of the requirement for approximate performance figures only, benchmark implementations were not carried out.

Figures 2.1, 2.2 and 2.3 give the graphs of inverse total computation times for the benchmarks. Computation times can be reduced by appropriate factors when the use of a faster CPU clock is assumed, for example the use of a 12MHz MC68000 instead of an 8MHz MC68000.

2.2.1.2 Analysis of Results

The following basic conclusions can be drawn from the benchmark results:-

(1) Performance comparisons across the class divisions. With or without the MPY-16 the general performance trend is:

(S(2) & S(1)) > S(0)-II > S(0)-I

The performance trend is much more pronounced for benchmarks without the MPY-16 due to the large variation in on-chip multiplication capability. When considering benchmark performance with the MPY-16, the general superiority of the S(1) and S(2) systems can be put

down to several interlocking factors:-

(a) Faster technology.

(b) Instruction pipelining.

(c) Higher levels of integration, so that wider data and address paths, sophisticated addressing, and a greater number of registers can be used.

The benchmark performance of S(1) and S(2) systems without the MPY-16, is superior to that of S(0) systems with the MPY-16 because of the above factors (improved on-chip multiplication capability being related to (c)). Note that the wider data paths of the S(1) and S(2) systems are less advantageous for BIII, as this benchmark requires only 8-bit arithmetic.

(2) Speed-up using the MPY-16.

As is to be expected the speed-up factor obtained assuming the use of an MPY-16, follows the trend:-

S(0)-I > S(0)-II > (S(1) & S(2))

For example the speed-up for Z80 benchmarks is in the range 3.4-8.0, whereas for MC68000 benchmarks the speed-up is only about 1.4. The greatest influence on this trend is the ratio of on-chip multiplication time to the time for other operations, for example register-memory moves, these ratios being much greater for S(0) systems. For S(1) and S(2) systems the three instructions needed to access the MPY-16 are not much faster than the 16 bit signed multiplication time, hence the marginal speed-up compared with S(0) systems.

(3) Useful Instructions.

The S(2) micros, the MC68000 and TMS9995 appear to have superior DSP performance overall. Other than faster multiplication, and faster technology in general, it is difficult to ascertain why this is true. The following gives some other features which appear to have reasonable influence on the observed performance trend:

(a) The use of on-chip registers or memory for holding temporary variables, therefore reducing costly external data memory accesses.

(b) The availability of at least three index registers, to save on register moves for benchmarks such as BII (which requires three indices for the butterfly routine).

(c) The fast autoincrement/autodecrement of address registers seemed to be useful for all three benchmarks.

(d) A fast instruction for decrementing a loop counter and testing for a branch condition.

2.2.2 Digital Signal Processing Micros

Applying the classification of section 2.1, the DSP micros studied can be conveniently split into two architectural classes:

S(-3): Vertically-biased RISC.
TMS32010, S2811, IBM RSP, (INTEL 2920).

S(-4): Horizontally-biased RISC.
BELL DSP, NEC 7720.

All of these DSP micros have separate data and program memory areas. Table 2.3 provides data on some of the DSP micros, see references [C-4] and [M-1] for further summaries.

Two basic arithmetic schemes are used, the adder based ALU (AB) and the multiplier-based ALU (MB), see [T-2] for more discussion on the issues involved. The latter scheme appears to be more popular with the later DSP micros, for example it is used in the NEC7720 and TMS32010. A major advantage of the MB scheme is that additional parallelism can be exploited, by having operations carried out in parallel with

multiplication i.e. by providing maximum support for the X.Y+A operation required for most digital filter tasks (X is the data operand, Y the coefficient, and A the accumulation). For example, the NEC7720 and S2811 use separate memory areas and separate buses for data and coefficients, and can execute the X.Y+A operation in a single cycle. The BELL DSP also executes this operation in a single cycle, but, by using multiplexed buses instead of separate buses.

Most DSP micros use functional hardware to support the sample delay operation, for which two basic schemes are found:-

(1) Physical movement of data.
(2) Adjustment of address pointers (Logical data movement).

The first scheme is the most popular, for example the TMS32010 and S2811 use special-purpose hardware for this operation, and the NEC7720 and BELL DSP support it, but with hardware which can be used in other operations i.e. utilising their ability to execute more than one micro-operation in a single cycle. In all four cases the sample delay operation can be carried out in parallel with the X.Y+A operation for greater digital filtering speed. An example of the second scheme is 'circular buffering', as used in the IBM RSP. This requires that data addresses in FIR and IIR filters be calculated modulo N (where N is the number of delay elements), such that no physical delay operations are required. Note that 'circular-buffering' is also useful for I/O buffering in applications such as overlap-add convolutions [O-1].

Other specific functional hardware of interest includes saturation logic, which can be used for eliminating large-scale limit cycles in 2's complement arithmetic when the number representation is exceeded [E-1,F-2]. This feature is used on all DSP micros to some extent, although being less fully supported on some DSP micros such as the NEC7720. Other functional features include the REPT instruction on the S2811 for FIR filters, and the barrel shifter on the TMS32010 used for scaling, bit manipulation and floating point arithmetic.

2.2.2.1 Performance

Unlike GP micros, data on the DSP performance of DSP micros is more readily available (exceptions being DSP micros developed for 'in-house' use, for example the IBM RSP and to some extent the BELL DSP). Table 2.3 gives collated DSP performance figures. These figures must be treated with caution due to the variation in implementation algorithms.

2.2.2.2 Analysis

This analysis considers digital filtering and FFT tasks only. The two DSP micros using Adder-based ALUs, the INTEL2920 and IBM RSP are not considered further, as the INTEL2920 is designed for a more limited range of applications (which does not include the FFT), and comprehensive data on the IBM RSP is unavailable.

Consider first the 32 tap FIR filter, for this benchmark the NEC7720 appears to be the fastest overall. This is because the NEC7720 performs the X.Y+A and sample delay operations in a single cycle, and also has a sufficiently fast cycle time i.e. the TMS32010 has a faster cycle time but less parallelism, and the S2811 and BELL DSP have the same parallelism as the NEC7720 but have slower cycle times.

For the bi-quad filter section benchmark, figures are too close to warrant further investigation, although it should be noted that when saturation arithmetic is required the NEC7720 will require additional computation overheads. Therefore nullifying some of the advantages obtained with the FIR benchmark.

Table 2.3 shows that the NEC7720 and TMS32010 have the best FFT performances. Before proceeding further with the FFT discussion, fundamental differences between these two devices should be elaborated on. The TMS32010 has vertically-biased RISC instructions and relies primarily on absolute addresses, this is in contrast to the NEC7720 which has horizontally-biased instructions and no absolute addresses. This effectively means that the NEC7720 whilst having hardware well adapted to some particular tasks (e.g. FIR filters), it is less flexible generally, being restricted by cumbersome addressing. The TMS32010 is not restricted in this way as will be described.

The FFT can be coded using either program loops or straight-line coding where, in general, the latter technique is faster as address calculation/movement and control overheads can be eliminated. Of the two DSP micros, the NEC7720 and TMS32010, only the TMS32010 can use straight-line coding as this technique requires the pre-calculation of absolute addresses and their use as part of the instruction. Because of the large program memory provided with the TMS32010 (compared with other DSP micros), straight-line coding of complex FFTs up to 64 points is possible, so that the TMS32010 is considerably faster than the NEC7720 over this range.

NOTE: General-purpose micros are also capable of using absolute addressing for straight-line sequences, but, because these are Von Neumann architectures the overheads in fetching pre-calculated address operands (as part of instructions) is much greater than for the TMS32010. Therefore the use of on-chip address arithmetic sequences is often faster.

For single-loop FFTs the TMS32010 can use pre-calculated offset addresses stored in program memory i.e. three offset addresses per FFT butterfly. Because the TMS32010 is a Harvard machine, the passing of values between program memory and the data area is more difficult than for a single memory architecture. This transfer requires a special Bus Interface Module (BIM), and the accessing of constants held in program memory takes 3 clock cycles, in contrast to the usual 1-2 clock cycles for instruction execution. For the actual

butterfly execution, both the TMS32010 and NEC7720 require the use of 'programmer look-ahead' [M-5] for the modification of indirect address values in parallel with data arithmetic. Execution times for the looped FFT for the NEC7720 and TMS32010 are approximately the same, so that the cumbersome addressing of the NEC7720 probably balances with the slow offset address and coefficient accessing of the TMS32010.

The FFT is a more demanding task than digital filtering, and therefore highlights DSP micro deficiencies more successfully. For example, it can be seen that data arithmetic cycles occur more frequently in digital filter tasks than in looped FFT programs, due partly to the general lack of fast sophisticated addressing in DSP micros.

2.3 Summary

The above sections consider GP micros and DSP micros as separate groups, we shall now briefly compare the DSP performance of GP micros with that of DSP micros. Consider the DSP performance of the fastest benchmarked general-purpose system i.e. MC68000 with an MPY-16, with that of the fastest DSP micro systems i.e. an TMS32010 or NEC7720, using similar implementation algorithms. For digital filter tasks, these DSP micros are approximately 20 times faster, and for small size FFTs (less than 64 points and non straight-line coding) these DSP micros are approximately 5 times faster than general-purpose systems.

This shows that the combination of RISC architecture with special functional hardware, can give dramatic DSP performance increases over general-purpose CISC architectures for tasks that fully exploit this functional hardware. But, the range of applications where this improvement applies is relatively narrow.

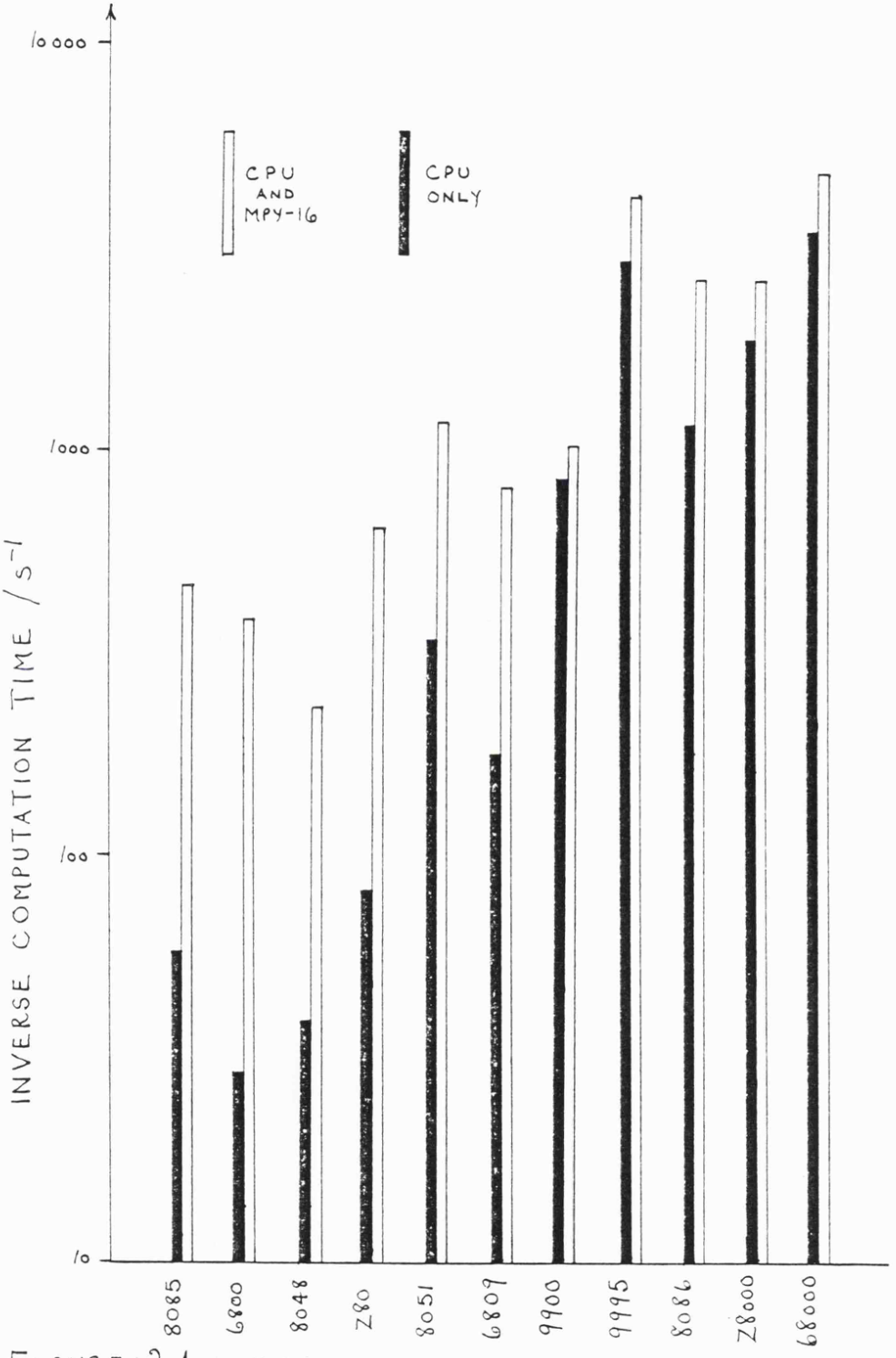


FIGURE 2.1. INVERSE COMPUTATION TIMES FOR BENCHMARK I.

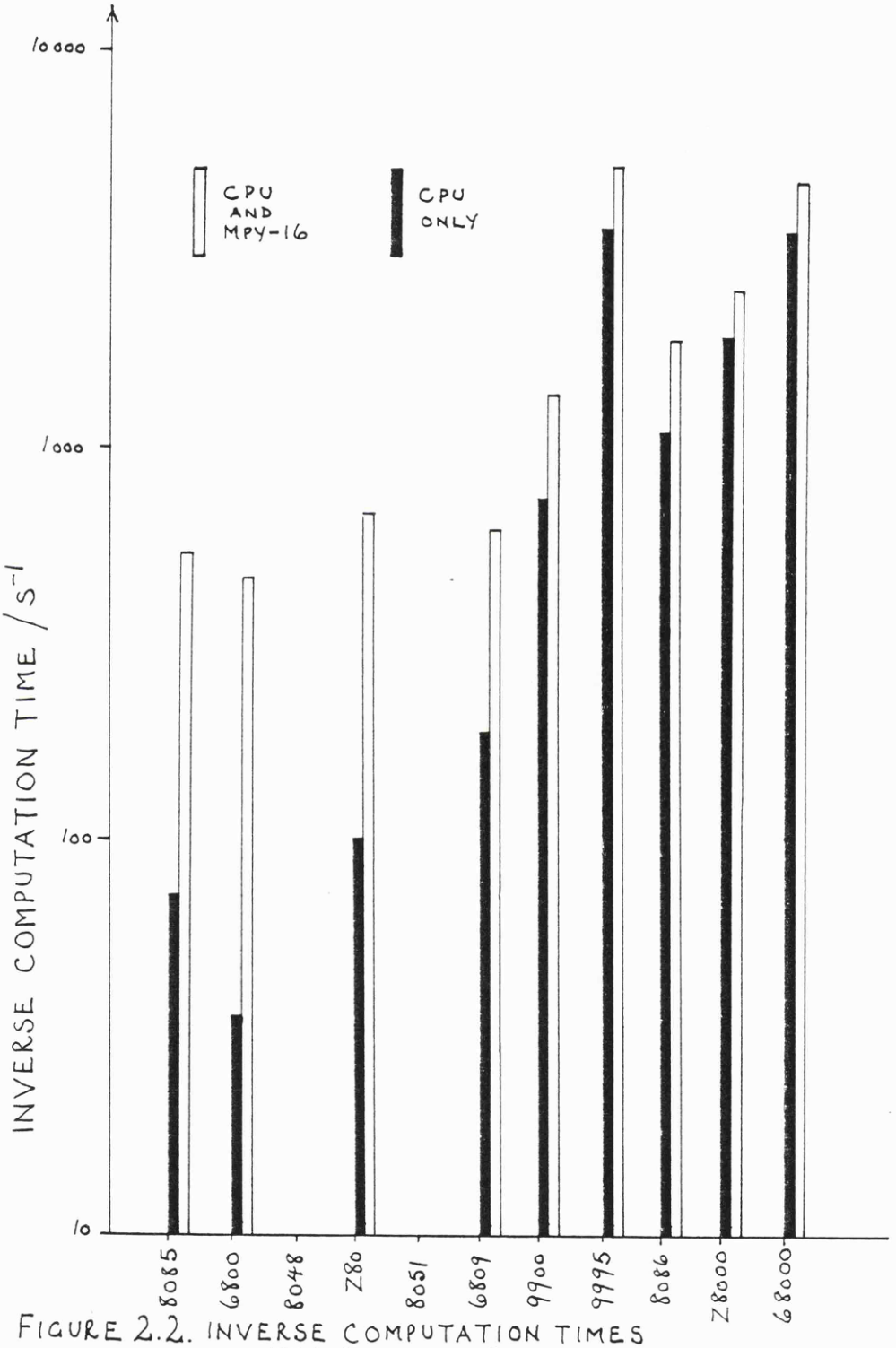


FIGURE 2.2. INVERSE COMPUTATION TIMES FOR BENCHMARK II

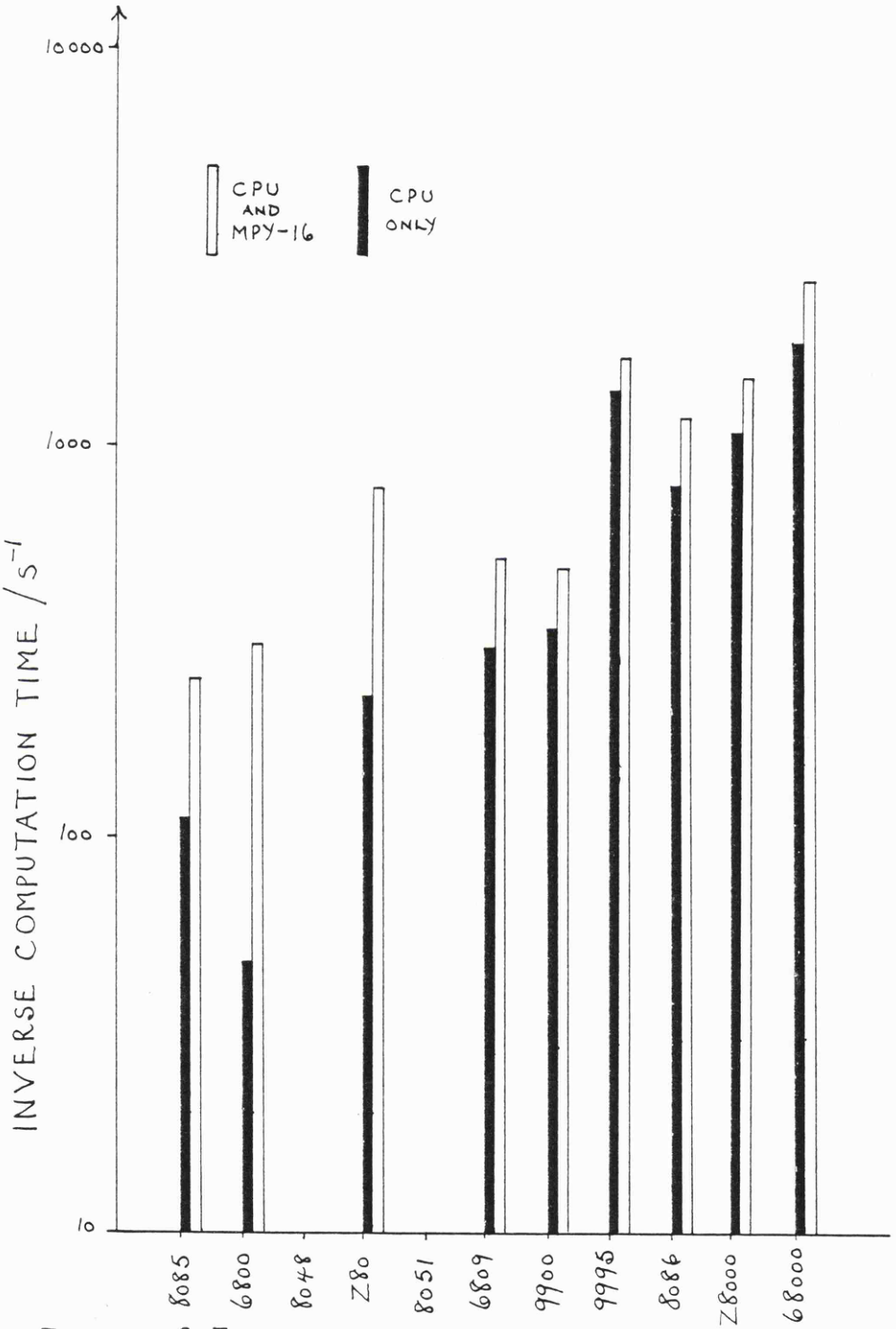


FIGURE 2.3. INVERSE COMPUTATION TIMES FOR BENCHMARK III

	IMAGE INSTRUCTION EXECUTION CONTROL	IMAGE INSTRUCTION TYPE	HOST INSTRUCTION TYPE	IMAGE MEMORY ARCHITECTURE	PIPELINE LEVELS	MULTIPLE ARITHMETIC UNITS ?	CLOSE /POSSIBLE EXAMPLES
S(-5)	RANDOM LOGIC	RISC	HORIZONTAL BIAS	SEPARATE MEMORY	≥ 2	YES	FDP [G-3]
S(-4)	RANDOM LOGIC	RISC	HORIZONTAL BIAS	SEPARATE MEMORY	≥ 2	NO	BELL DSP [B-5]
S(-3)	RANDOM LOGIC	RISC	VERTICAL BIAS	SEPARATE MEMORY	≥ 1	NO	IBM RSP [M-2]
S(-2)	RANDOM LOGIC	RISC	VERTICAL BIAS	VON NEUMANN	≥ 1	NO	RISC I [P-1]
S(-1)	RANDOM LOGIC	RISC -BIASED	VERTICAL BIAS	VON NEUMANN	1	NO	T424 TRANSPUTER [B-8]
S (0)	RANDOM LOGIC	CISC -BIASED	VERTICAL BIAS	VON NEUMANN	0	NO	MC6800
S (1)	RANDOM LOGIC	CISC	VERTICAL BIAS	VON NEUMANN	1	NO	Z8000
S (2)	2-LEVEL MICROCODE	CISC	VERTICAL BIAS	VON NEUMANN	1	NO	MC68000
S (3)	1-LEVEL MICROCODE	CISC	HORIZONTAL BIAS	VON NEUMANN	1	NO	HP MICRO [G-2]
S (4)	1-LEVEL MICROCODE	CISC	HORIZONTAL BIAS	VON NEUMANN	≥ 2	NO	AMP [B-4]
S (5)	1-LEVEL MICROCODE	CISC	HORIZONTAL BIAS	VON NEUMANN	≥ 2	YES	TI ASC [B-10]

TABLE 2.1. S-TYPE CLASSIFICATION

CLASS	MICRO	CLOCK FREQUENCY	INTERNAL BUS	EXTERNAL BUS	ON-CHIP MEMORY	MULTIPLICATION		
						INSTN.	16 BIT SIGNED	8-BIT SIGNED
S(0)-I	I 8085A	3MHz	8	8	N	——	750μs	176μs
	MC 6800	1MHz	8	8	N	——	1651μs	550μs
	I 8048	400kHz	8	8	Y	——	1050μs	240μs
	Z 80	4MHz	8	8	N	——	550μs	125μs
	I 8051	12MHz	8	8	Y	8-BIT UNSIGNED	97μs	21μs
S(0)-II	MC 6809	1MHz	16	8	N	8-BIT UNSIGNED	218μs	59μs
	TMS 9900	3MHz	16	16	N	16-BIT UNSIGNED	60μs	60μs
S(1)	Z 8000	4MHz	16	16	N	16-BIT SIGNED	18μs	18μs
S(2)	I 8086	3MHz	16	16	N	16-BIT SIGNED	34μs	21.8μs
	TMS 9995	3MHz	16	16	Y	16-BIT SIGNED	9.3μs	9.3μs
	MC 68000	8MHz	16	16	N	16-BIT SIGNED	9.25μs	9.25μs

TABLE 2.2. DATA ON GENERAL-PURPOSE MICROS BENCHMARKED.

		Intel 2920	NEC 7720	AMI S2811	TMS32010 Microcomputer	Bell DSP
Program memory		192 × 24	512 × 23	250 × 17	1536 × 16	512 × 23
Data Memory	Ram	40 × 25	128 × 16	128 × 16	144 × 16	128 × 16
	Rom	—	512 × 13	128 × 16	—	512 × 13
Hardware Multiplier		—	16 × 16 = 31	12 × 12 = 16	16 × 16 = 32	20 × 16 = 36
cycle time		400 nS	250 nS	300 nS	200 nS	800 nS
32 tap FIR		38·4 μsec.	9·25 μsec.	10·8 μsec.	12·8 μsec.	31 μsec.
Biquad section		5·6 μsec. (approx.)	2·25 μsec.	2·1 μsec.	2·2 μsec.	3·2 μsec.
Complex FFT	32 pt.	—	700 μsec.	1500 μsec.	291 μsec.	N/A
	64 pt.	—	1600 μsec.	N/A	737·6 μsec.	N/A
Type		S (-3).	S (-4)	S (-3)	S (-3)	S (-4)

TABLE 2.3. DATA ON DSP MICROS STUDIED

3. THE MAC68 - A FUNCTIONAL MULTI-PROCESSOR SYSTEM.

The MAC68 system was designed as a result of the analyses of chapter 2, the objective being to boost the DSP performance of a general-purpose micro to at least the level of the fastest available DSP micro, but to do this using the minimum of special-purpose architectural features. A system based on a General-purpose micro has many advantages over it's special-purpose counterparts, for example:

(i) Inherent flexibility.

GP micros can be programmed for most applications, this being their primary design criterion.

(ii) Hardware support.

Most GP micros are supported by a range of input-output peripherals, memory devices etc., Consequently, systems can be expanded to meet new requirements.

(iii) Software support.

Systems software *eg.* compilers, assemblers, debuggers, operating systems etc., are available for most popular General-Purpose micros. The techniques of generating and using program code for GP micro architectures are now well established.

If these advantages can be retained without losing out in DSP performance terms, then there is potential for maximising the DSP application range and easing development problems at the same time.

As chapter 2 demonstrates, the increase in speed obtained for S(2) General-purpose micros by using an external multiplier, is not sufficient to approach DSP micro performance when considering DSP tasks. Therefore a Multiple-Instruction (MI) stream solution could be sought i.e. a Functional MI stream (FM) solution or Modular MI stream (MM) solution (see chapter 1 for definitions). For a Modular MI stream system, even when assuming sufficient algorithm parallelism and zero inter-processor communication overheads, at least 20 General-purpose micros would be required to approach the performance of a TMS32010 when considering basic digital filtering tasks. Consequently, Modular MI stream systems were rejected in favour of the Functional MI stream approach. Consider then the following Classification for FM systems:

FM(1) = S(1)	Single-instruction stream only.
FM(2)	Two SI stream processors. e.g. Thirion's system [T-3]. MC68000 + MC6881 co-processor.
FM(3)	Three SI stream processors. e.g. SPS-41/81 [F-3].
FM(>3)	Greater than three SI stream processors. e.g. ST-100 [H-2]. SPERRY system [H-1].

This classification is based on the number of functional Single-instruction stream processors used, this being the best indication of parallelism ('n' in F(n) indicates the number of processors). The most extreme examples of Functional MI stream systems are the 'array processors', for example the ST- 100 which uses at least 3 special-purpose processors. Array processors are specially designed to process large blocks of data efficiently, and use a large amount of special-purpose hardware which cannot be justified for more flexible architectures. We shall therefore only consider F(2) systems (2 processors), where one of the processors is a General-purpose micro.

LSI/VLSI functional processors available which could enhance the DSP performance of a GP micro are the Floating-Point Co-processors and of course DSP micros. For the former case, reported examples of DSP implementations are not encouraging [S-2] mainly because of the orientation of Co-processors towards improving the floating-point arithmetic performance of GP micro systems, not fixed-point arithmetic performance. In the latter case, the migration of most of the DSP functions to the DSP micro does not fit our established design criterion. Finally, it was decided that the best solution was to design a custom arithmetic processor which could co-operate with a GP micro. This custom processor would make efficient use of one of the available high-speed fixed-point arithmetic devices i.e. much more efficient use than was assumed with the MPY-16 in the GP micro benchmarks.

3.1 Architecture and Programming of the MAC68 System

The MAC68 uses two Single-instruction stream processors, one is a General-purpose 16-bit microprocessor, the MC68000, and the other is a custom-built arithmetic processor, the MACDSP, which is built around the TRW TDC1010J multiplier-accumulator.

The architecture of the MAC68 has been designed so that both processors can execute tasks in parallel, the MC68000 can execute tasks such as index address calculation, input-output processing, iteration control and complete background tasks, while the MACDSP executes sequences of arithmetic. This parallelism is made possible by using double-buffers between the two processors. Architectural details were decided on after studying the requirements of basic DSP tasks, in particular digital filtering, spectral analysis and correlation.

The MC68000 (8MHz) was chosen as the General-purpose processor for the system, because it had superior overall performance in the Chapter 2 benchmarks. For the custom arithmetic processor, the TRW TDC1010J multiplier-accumulator is the most suitable building block, because it can execute the X.Y+A operation in 155nsec (maximum). The importance of the X.Y+A operation was discussed in Section 2.2.2.

The distribution of functions in the MAC68 is similar to that found in Thirion's system [T-3], and the inter-processor parameter passing used is similar to that of the SPS-41/81 [F-3], although the particular combination of features in the MAC68 and the use of up-to-date technology makes the MAC68 architecture novel.

3.1.1 Overview

Figure 3.1 gives a pictorial view of the complete MAC68 hardware system which includes the MC68000 system components, the MACDSP and the Analog Input-Output (I/O) sub-system. The majority of the hardware for the MC68000 system components is contained within the SAGE II microcomputer system, and therefore did not have to be developed specifically for this study. The remainder of the MAC68 hardware is contained in two circuit boards, the MACDSP board using 60 ICs and the Analog I/O sub-system using 19 ICs. Software provided with the SAGE II system was used in the development of the MAC68 support software, and in the development of MAC68 application software.

The 16-bit MC68000 microprocessor is an advanced CISC-type processor, using instruction pipelining, sophisticated addressing modes, 16 general-purpose 32-bit registers and a variety of interface mechanisms which include an 8-level priority interrupt scheme. Minimum execution times are obtained by using 0-Wait state memory for the MC68000.

The MC68000 services both the MACDSP and the Analog I/O, which are assigned interrupt priorities and interrupt the MC68000 to synchronise activities. These activities may occur in parallel.

The MACDSP custom arithmetic processor uses a 3-stage synchronous pipeline, with all instructions executing in a single 250nsec clock cycle.

Programs, constants and coefficients are loaded into the MACDSP program memory by the MC68000, therefore easing program development considerably. To execute MACDSP programs, the MC68000 firstly provides the MACDSP with data, which is loaded into the MACDSP data memory, the MC68000 then loads a start address and offset addresses into the MACDSP and then initiates the execution of a MACDSP program sequence. The offset addresses are added to the main addresses (which are a field of each MACDSP instruction) to give effective indexed addressing, this feature is particularly useful for circular buffering and for FFT indexing. Other fields in the MACDSP instruction are used for controlling data arithmetic and data movement. High arithmetic speed is possible because MACDSP arithmetic operations, for example multiplication, execute in a single 250nsec cycle, each cycle fully overlapping instruction processing, data accessing and data arithmetic. The MACDSP uses a simple non-branching program control scheme, whereby, the MC68000 initiates execution and a MACDSP instruction field is used to terminate MACDSP execution, sending an interrupt signal to the MC68000 as part of the process.

Because the start address latch and offset registers on the MACDSP are double-buffered, the MC68000 can prepare and load a set of parameters (start address and offset addresses) while the MACDSP is executing a program sequence using the current set of parameters. This means that for some applications, separate MACDSP program sequences can be effectively strung end to end, with the MACDSP continuously active. Efficient use of this feature is only possible where the MC68000 and MACDSP processing loads are balanced, and in some circumstances MACDSP sequences may have to be extended to achieve this balance (this method relies on the use of an absolute address field, and is similar to the straight-line coding described in Chapter 2 for the TMS320).

Support software developed specifically for the MACDSP includes an assembler, debugger and loader.

The main devices on the Analog I/O sub-system are, a 12-bit Analog-to-Digital convertor (ADC), a 12-bit Digital-to-Analog convertor (DAC) and a Sample-Hold (S/H). The MC68000 initiates real-time processing by starting a counter in the Analog I/O sub-system, this counter provides a pulse every sample period whch starts A/D and D/A conversion processes. On completion of the A/D conversion process, the MC68000 is interrupted and I/O processing is carried out. For applications requiring the input-output of blocks of data samples, MC68000 I/O processing can be carried out in parallel with MACDSP arithmetic.

In Appendices II-IV low-level descriptions are given of MAC68 logic, and in Appendices V-VII descriptions are given of MAC68 support software. The following concentrates on higher-level details and design philosophies.

3.1.2 MACDSP

Figure 3.2 gives a detailed illustration of MACDSP architectural features.

3.1.2.1 Major Design Features

The main MACDSP design objective was to provide an interface between the MC68000 and the TRW TDC1010J such that both devices can be used efficiently. Use of double-buffering in the MACDSP - MC68000 interface helps to ensure that both processors can execute in parallel. The following describes major MACDSP design features pertaining to the efficient use of the TRW TDC1010J.

The MACDSP uses a 3 stage synchronous pipeline, with buffering between each stage. The three pipeline stages function in parallel, and are:

(1) Instruction processing.

This stage involves first the incrementing of the program counter to give a new instruction address. Using this address an instruction is fetched, and the absolute address part of the instruction is added to a selected offset register value to form the data address. Both

this data address and the control bits of the MACDSP instruction are then clocked into registers, ready for the next stage in the pipeline.

(2) Data movement.

Using the data address and some of the control bits which are all held in the pipeline registers, a single data operand can be moved between the data RAM and the MAC. When moving a data operand from the data RAM to a MAC input register, the input register acts as the pipeline buffer for the next stage.

(3) Data arithmetic.

An arithmetic operation is carried out on the values held in the MAC input registers and on the value held in the MAC accumulator. The result is clocked into the MAC accumulator.

Note that complete overlap of data and instruction accessing is possible because of the use of separate memories for data and instructions (as with a Harvard machine). Another advantage of using separate memories, is that instruction sizes do not have to conform to particular data sizes.

As with all processors using synchronous pipelines, the pipeline clock cycle (MACDSP clock cycle) must be sufficiently long to account for the longest of the pipeline activities. This prototype MACDSP uses a clock frequency of 4MHz, as this frequency is easily derived from the MC68000 clock, and because fairly wide tolerances were required in case of major design changes during MACDSP development. The longest executing MACDSP pipeline activity, the data arithmetic activity, takes approximately 200nsecs.

It was decided that a single data transfer per MACDSP instruction cycle would be sufficient for this MACDSP prototype, although the advantages of having two data transfers per instruction cycle for digital filtering tasks have already been noted (see Chapter 2). Two data transfers per cycle could be accomplished either by using two data memory areas or by multiplexing the data and address buses connected to a single data memory area. Data RAM word size is 16 bits to conform to the 16 bit operations used in the MAC, and the number of data words is 1024, this was thought sufficient for most applications under consideration.

All of MACDSP program memory is RAM, and is loaded up by the MC68000. Up to 1024 MACDSP instructions can be held in the MACDSP program RAM each instruction being 24 bits long. With this many instructions, MACDSP program sequences of sufficient length can be developed to ensure efficient use of the self-start mode (to be described later). Instructions are horizontal, so that each control bit directly controls a MACDSP resource. This is slightly inefficient in terms of instruction memory usage, as some control bit combinations may never be used, and instruction encoding could be used. NOTE: Two instruction bits are unused anyway, adding further to this inefficiency (see Chapter 5 for potential enhancements).

3.1.2.2 MACDSP Instructions and Execution

Figure 3.3 gives the syntax for MACDSP assembly programs, which consist of three basic sections:-

(1) Data constants.

(2) Coefficients.

(3) MACDSP program sequence.

Prior to real-time execution, the MC68000 loads constants and coefficients into MACDSP data memory, and MACDSP program sequences into MACDSP program memory.

NOTE: Any data or instructions could be altered by the MC68000 during real-time processing if required by an application.

Once the MC68000 has loaded a start address and offset address values into the MACDSP, a start pulse is then sent to the MACDSP. Assuming that the MACDSP is not already active, this has the effect of initiating a MACDSP program sequence. Then the start address becomes the instruction address on the first cycle, and the offset register banks are switched so that the offset addresses loaded into the MACDSP prior to the sending of the start pulse, become the active offset addresses during the initiated program sequence. The use of switched register banks in this way gives effective double-buffering.

The start address latched into the program counter during the first cycle, is automatically incremented at the beginning of each cycle for the duration of the program sequence.

While the MACDSP is active, the MC68000 data bus is effectively disconnected from the MACDSP internal data and instruction buses, and the MC68000 address bus is effectively disconnected from the MACDSP data address bus. This also has the effect of disconnecting the MACDSP instruction bus from the MACDSP data bus. These disconnections avoid bus conflicts, and are carried out by putting the relevant 3-state buffers at a high impedance. Consider now the details of actual MACDSP instructions. The following gives the names and positions of MACDSP instruction bits.

0 - R/W.

1 - MAC OP.

2 - CLK X.

3 - CLK Y.

4 - END.

5 - CLK P.

6 - SUB.

7 - ACC.

8 & 9 - OFFREG. SELECT.

10 & 11 - UNUSED.

12-23 - MAIN ADDRESS.

There are three main fields:-

(1) Offset register select field (bits 8&9).

Two banks of four offset registers are used, each register being 12 bits long. Only one register bank can be read during any particular MACDSP program sequence. An offset register is selected on every cycle using the offset register select bits, and the contents of the selected register are added to the main address to give the index address for the required data operand. For programs containing instructions that do not require indexing i.e. absolute addressing only, one of the offset registers must be loaded with zero (and in many cases this will have to be done for an offset register in both register banks).

(2) Main address field (bits 12-23).

All MACDSP data addresses and address paths are 12 bits long, even though the data memory is only 1Kwords. This is a consequence of the logic used being available only as multiples of 4 bits, although it does mean that MACDSP data memory can be extended to 4Kwords with fewer significant hardware changes. The advantages of using an absolute address field will be discussed in section 3.2.

(3) Control field (bits 0-7).

The control field is used for the following activities:-

(a) Controlling the reading and writing of the MACDSP data RAM by enabling or disabling the write pulse.

(b) Enabling clocks to any of the MAC registers.

(c) Enabling the MAC output.

(d) Controlling MAC arithmetic operations, by providing a 2-bit MAC instruction on every cycle.

(e) Terminating a MACDSP sequence.

Before giving examples of instructions, we must discuss MAC arithmetic in more detail. Data arithmetic is carried out on the contents of 3 internal MAC registers:-

X input register (16 bits)

Y input register (16 bits)

ACC accumulator (35 bits)

Data arithmetic operations allowed are:-

Multiplication	X.Y
Multiplication-accumulation	X.Y+ACC
Multiplication-subtraction	X.Y-ACC

To allow coefficients to be in the range -2<..<+2, and data to be in the range -1<..<+1, additional hardware has been incorporated in the MACDSP design. For example, if a coefficient in the range -2<..<+2 is loaded into the X register, a data value in the range -1<..<+1 loaded into the Y register and arithmetic performed on these values, then the range of the arithmetic result stored back to Data RAM will be -1<..<+1. The advantage of this feature is twofold, in that copefficients for bi-quad filter sections can be represented without scaling, and exact +1 and exact -1 coefficients can be used for addition and subtraction operations. The major disadvantages of this feature are that coefficients are represented with reduced resolution, and that values stored to data RAM from the MAC must be truncated as opposed to

being rounded (rounding can only be used with the MAC when a specific range of the MAC accumulator is used for the MAC output - which it is not for this configuration. See the TRW TDC1010J data sheet for further details).

Saturation logic is not implemented on the MACDSP, and shifts must be carried out using a full multiplication, although neither of these seriously hinders DSP arithmetic.

The following MACDSP assembly program illustrates the fundamentals of MACDSP programming (see Figure 3.3 for syntax and Appendix V for semantics):-

```
.D
      500, 0.2 ; DATA RAM LOCATION 500 IS SET-UP WITH A VALUE
               ; 0.2 PRIOR TO SEQUENCE EXECUTION.

.C
      501, -1.6; DATA RAM LOCATION 501 IS SET-UP WITH A
               ; COEFFICIENT VALUE OF -1.6 PRIOR TO SEQUENCE
               ; EXECUTION.

.P
      S750     ; FIRST MACDSP INSTRUCTION AT PROGRAM RAM
               ; LOCATION 750.
               ; ASSUME:
               ; OFFSET REGISTER 1=0
               ; OFFSET REGISTER 2=200
               ; OFFSET REGISTER 3=100
```

```
X2D,300     ; LOAD X REGISTER WITH THE CONTENTS OF DATA
            ; RAM LOCATION 200+300=500.

Y3M,401     ; LOAD Y REGISTER WITH THE CONTENTS OF DATA
            ; RAM LOCATION 100+401=501, AND MULTIPLY
            ; X AND Y TOGETHER.

N1D,0       ; PIPELINE DELAY, ARITHMETIC IS CARRIED
            ; OUT DURING THIS CYCLE.

P1D,502     ; THE ARITHMETIC RESULT, HELD IN THE ACCUMULATOR,
            ; IS STORED TO DATA RAM LOCATION 502+0=502.
            ; IT'S VALUE WILL BE -0.32.  THE END BIT
            ; WILL BE ACTIVE FOR THIS INSTRUCTION.

.E
```

A no-operation instruction is needed in the above example because of the pipeline delay. In practice, this pipeline delay period can usually be usefully employed.

A MACDSP program sequence is terminated when an instruction containing an active END bit is executed. For the above example the assembler would ensure that only the last instruction in the sequence has the END bit active.

Because the logic enabling the MACDSP start sequence is double-buffered, 'self-start' can be used, whereby as soon as a MACDSP sequence terminates, it will immediately start up again if the buffered start logic has been set by the MC68000. The actual start-up sequence is exactly the same as described previously, so that a new start address and new offset addresses can be used. If required the old start address can be reused without requiring the MC68000 to perform another load. Correspondingly, for the offset registers, if an offset address value must remain constant throughout a series of 'self-starts', the relevant offset address value need only be loaded into the same register position for both register banks i.e. bank switching will not affect the output from this register position.

3.1.2.3 MACDSP Program Testing

Hardware is provided on the MACDSP so that between the execution of separate MACDSP program sequences, values held in all MAC resisters remain unchanged. This feature is used for testing MACDSP programs, as all MACDSP instructions can be loaded with their END bits set active, so that MACDSP program sequences can be effectively single-stepped by using a start address which is modified to point to each instruction in a sequence.

A debugger software package has been developed to support this method (see Appendix VI). The debugger single-steps program sequences, displaying the contents of specific data RAM locations and the MAC accumulator, where required in the sequence.

3.1.3 MC68000 Processing Activities

SAGE II software/firmware used for the development of MC68000 application programs includes, a MC68000 assembler and a code debugger. A MC68000 code loader was developed specifically for the MAC68 system.

The MAC68 system utilises the autovectoring capability of the MC68000 to synchronise the independent concurrent program activities. Autovectoring involves the execution of an Interrupt Service Routine (ISR) in response to an external interrupt. The autovector is the start address of the ISR and is held in MC68000 system RAM, whereby each autovector corresponds to a particular interrupt level.

During the autovectoring sequence the MC68000 program counter and status register are stacked, effectively storing the processor state usually being restored at the end of the ISR via an RTE instruction. Every time an ISR must be used this overhead is incurred, with the autovector sequence requiring 44 cycles to execute and the RTE instruction requiring 20 cycles to execute. This overhead is the most optimistic in that it assumes that register values will not have to be stored and restored as well as the program counter and status register (in practice this additional overhead can be avoided by assigning registers for the exclusive use of ISRs). In the MAC68 two autovector interrupt levels are reserved (each with a separate ISR):

Autovector level 3: Analog I/O sub-system.

Autovector level 2: MACDSP.

The Analog I/O is given a higher priority, as I/O servicing is generally a more critical activity than MACDSP servicing. Priority levels lower than level 2 are used for less time critical background tasks.

Note that when an interrupt of priority higher than the current priority is required, a process switch can only occur once the current instruction executing has terminated.

The following now discusses the specialised tasks performed by the MC68000. Non-specialised tasks such as background processing are application dependent, and are discussed in more detail in later sections.

3.1.3.1 I/O Servicing

The MC68000 initiates I/O processing by starting the I/O counter. This counter is hardwired to generate a pulse every 125usec (appropriate for the sampling of speech signals), which is used within the I/O sub-system to initiate both an analog-to-digital conversion and a digital-to-analog conversion. By using the same pulse for both conversions input-output timing is guaranteed.

Analog-to-digital conversion is performed on a value held in the sample-hold device, and when this conversion is complete, an interrupt (priority 3) is sent to the MC68000 and the Sample-hold device is put into sample mode. MC68000 response to the priority 3 interrupt is to firstly reset the I/O interrupt logic, and then to jump to the I/O ISR. Both A/D and D/A devices contain addressable latches, so that the I/O ISR normally fetches a 12 bit value from the A/D and sends a 12 bit value to the D/A. Other tasks carried out by the I/O ISR might include bipolar-2's complement and 2's complement bipolar conversions, data block management, setting of flags for synchronising concurrent programs etc..

Once the I/O counter has been started, the I/O ISR will be executed once every 125usec period.

3.1.3.2 MACDSP Servicing

Once the I/O sub-system has been activated, MC68000 servicing of the MACDSP consists of the following activities:-

(1) Accessing of the MACDSP data RAM.

(2) Loading MACDSP start addresses.

(3) Loading MACDSP offset addresses.

(4) Setting the MACDSP start logic.

The MACDSP signals the termination of a program sequence, by sending a level 2 interrupt signal. The MC68000 will respond to this interrupt on completion of the currently executing instruction, unless an I/O ISR is being executed on a higher priority level, in which case the interrupt will remain active until the I/O ISR has been completed. When the MC68000 does respond to the MACDSP interrupt request, the MACDSP interrupt logic is reset, and the MACDSP ISR executed.

The MACDSP self-start mode already described, is most effective where application algorithms are of the correct form, for example algorithms which involve the repeat of a basic computational structure, and require no access of MACDSP data memory between repeated executions. Greatest efficiency then, is obtained when the MC68000 and MACDSP processing loads balance.

Unfortunately, a difficulty can arise in using the MACDSP self-start mode. Assume for example the MACDSP is using the self-start mode, and has set the interrupt logic signalling the termination of a MACDSP program sequence and immediately re-started the execution of the next sequence. If the MC68000 does not reset the MACDSP interrupt logic (via the autovector sequence for the MACDSP ISR) before the termination of this next program sequence, then the first MACDSP interrupt signal will be effectively lost. The effect of this is to take the MACDSP out of self-start mode, such that the MACDSP ISR executes sequentially with MACDSP processing, instead of in parallel. The net effect is to increase the total computational time, although processing is still logically correct. This situation can be avoided by developing concurrent programs to fit timing constraints (see chapter 4). Chapter 5 suggests a general solution to this problem.

3.1.4 Setting-Up Real-Time Execution

A software package has been developed for the MAC68 which simplifies the setting up of a MAC68 real-time program environment (see Appendix VII). This package performs the following operations:-

(1) The loading of MACDSP code files into MACDSP memory areas.

(2) The loading of MC68000 code files into MC68000 memory (each of the MC68000 activities is assigned a separate area of memory).

(3) The loading of ISR start addresses into the MC68000 vector table.

(4) The initiation of real-time processing by passing control to an initialising routine (this routine usually starts the I/O counter, and then waits for the I/O ISR to start up concurrent activities).

3.2 Examples

Rarely is there a unique implementation of a DSP structure for a particular DSC, in the MAC68 case, various implementation alternatives exist even for the most common DSP structures.

The following explores the implementation of some of these common DSP structures on the MAC68. Particular emphasis is placed on the MAC68 use of straight-line sequences for reduced computation times.

3.2.1 Finite Impulse Response (FIR) Filters

The sample delays of FIR filters can be implemented with or without the use of circular buffers when using the MAC68. Non-circular buffering uses physical data moves to implement the sample delays, and requires 4 MACDSP instruction cycles for each tap used. In contrast, circular buffering requires only 2 MACDSP instruction cycles for each tap, although this improvement is offset by the time needed to comute offset addresses. The following details the use of the circular-buffer method, with only the FIR data arithmetic implemented as a straight-line MACDSP sequence.

In implementing the FIR data arithmetic as a single MACDSP sequence, only one MACDSP interrupt is required each time the FIR filter is executed (see chapter 5 for an alternative method). Unfortunately then, the number of MACDSP data memory locations used for data is double that required for non-circular buffering. The reason for this is illustrated in Figure 3.4.

Note that with this implementation method there is a minimum computation time. This is because offset address calculation and I/O carried out by the MC68000 in parallel with MACDSP computation (or in series), is independent of the size of FIR filter used i.e. this overhead cannot be eliminated by reducing the number of filter taps. The minimum computation time is approximately 6usec, such that all FIR filters (using circular buffers) of 24 taps and under, execute in this time.

Consider as an example, the maximum sampling frequency for an 128 tap FIR filter. Assuming the use of block I/O (with a block length of 256 samples) which is generally more efficient than stream I/O on the MAC68, and the use of the self-start mode. Then, the MACDSP can be made to execute continually except for when data must be moved to and from the MACDSP data memory every 256 sample periods. With this technique, offset address calculation and I/O could occur in parallel with data arithmetic, and a sample frequency of approximately 16kHz could be obtained.

NOTE: Circular-buffering and block I/O can also be used with Infinite Impulse Response (IIR) filters. Although circular-buffering is generally less successful with IIR computational structures.

3.2.2 Fast Fourier Transform (FFT)

Efficient implementation of the FFT on the MAC68 requires the use of straight-line butterfly sequences, such that the execution load of the MC68000 and the MACDSP is balanced.

Consider for example a 256-point complex FFT using the computational structure outlined in Benchmark II (of Appendix I), except with the first three passes of the FFT requiring only complex addition and subtraction i.e.

taking advantage of the need for only purely real or purely imaginary coefficients for these three passes. The resultant implementation on the MAC68 could have the following characteristics:-

(a) MACDSP Program Memory Usage for the FFT.

Four MACDSP program sequences could be used, corresponding to each of the first three passes and a fourth sequence for passes 4, 5, 6, 7 and 8. The program sequences for the first three passes could consist each of a straight-line sequence of 8 butterflies, each butterfly requiring 14 instructions. The fourth MACDSP program sequence also using 8 butterflies, would use 23 instructions for each butterfly. Figure 3.5 shows the MACDSP instructions required for the 23 instruction butterfly.

(b) FFT Execution.

For the first 3 passes only two of the offset registers would be required. One of these registers would hold zero, and the other register would be used for indexing complex data points. For the remaining passes all four offset registers would be required, two for indexing complex data points, one for indexing coefficients and the remaining offset register for holding zero. With this particular FFT implementation, the minimum computation time for any MACDSP program sequence will be 28usec, therefore the self-start mode could easily be used for the duration of the FFT computation. Therefore the

total number of MACDSP cycles required will be 20,672, corresponding to an execution time of 5.168msec.

3.2.3 Cross-Correlation

For large size FFTs or cross-correlations it is not possible to implement entire computational structures using straight-line sequences only. As it is desirable to implement these algorithms with continuous MACDSP operation (i.e. self-start mode), sequences must be partially expanded up to the point where this continuous operation is possible.

For cross-correlation, a novel technique can be used to reduce computation time, whereby start addresses for a MACDSP sequence do not always point to the first instruction in a MACDSP sequence, and where a value must be held in the MAC accumulator between MACDSP sequence executions.

The computational structure for cross-correlation is given as Benchmark II of Appendix I. The proposed MAC68 implementation has the following features:-

(a) MACDSP Program Memory Usage for Cross-Correlation. Three MACDSP program sequences can be used, each performing straight-forward sum-of-products calculations. Two of these sequences are used for the straight-line calculation of the correlation values which require less than 4 (say) product-sums in their calculation. The other MACDSP program sequence therefore represents the calculation of the sum

of N-4 products (where N is the number of data words in each input vector), the storage of the result and the zeroing of the MAC accumulator (at the end of the sequence).

(b) Cross-Correlation Execution.

One of the product sum sequences requires the calculation of offset addresses and of start addresses. The first start address used with this sequence points to the first instruction, and each subsequent start address will be two greater than the old start address (i.e. as each product requires two MACDSP instructions). The MAC accumulator is zeroed at the end of each execution sequence so that the following MACDSP execution sequence can start on an instruction which multiply-accumulates. All four offset registers must be used, two for indexing both operands in product calculation, and one for indexing the location where the correlation value is to be stored (the fourth register holds zero). The entire cross-correlation computational structure could be executed using the self-start mode if N is sufficiently large.

NOTE: This use of maintaining values in the MAC between MACDSP sequences is not fully explored for the above examples, as MAC68 interrupt overheads prevent computational structures from being split into several MACDSP sequences and efficiently executed separately. In chapter 5, details are given on a method for making this 'computational structure spliting' technique more efficient.

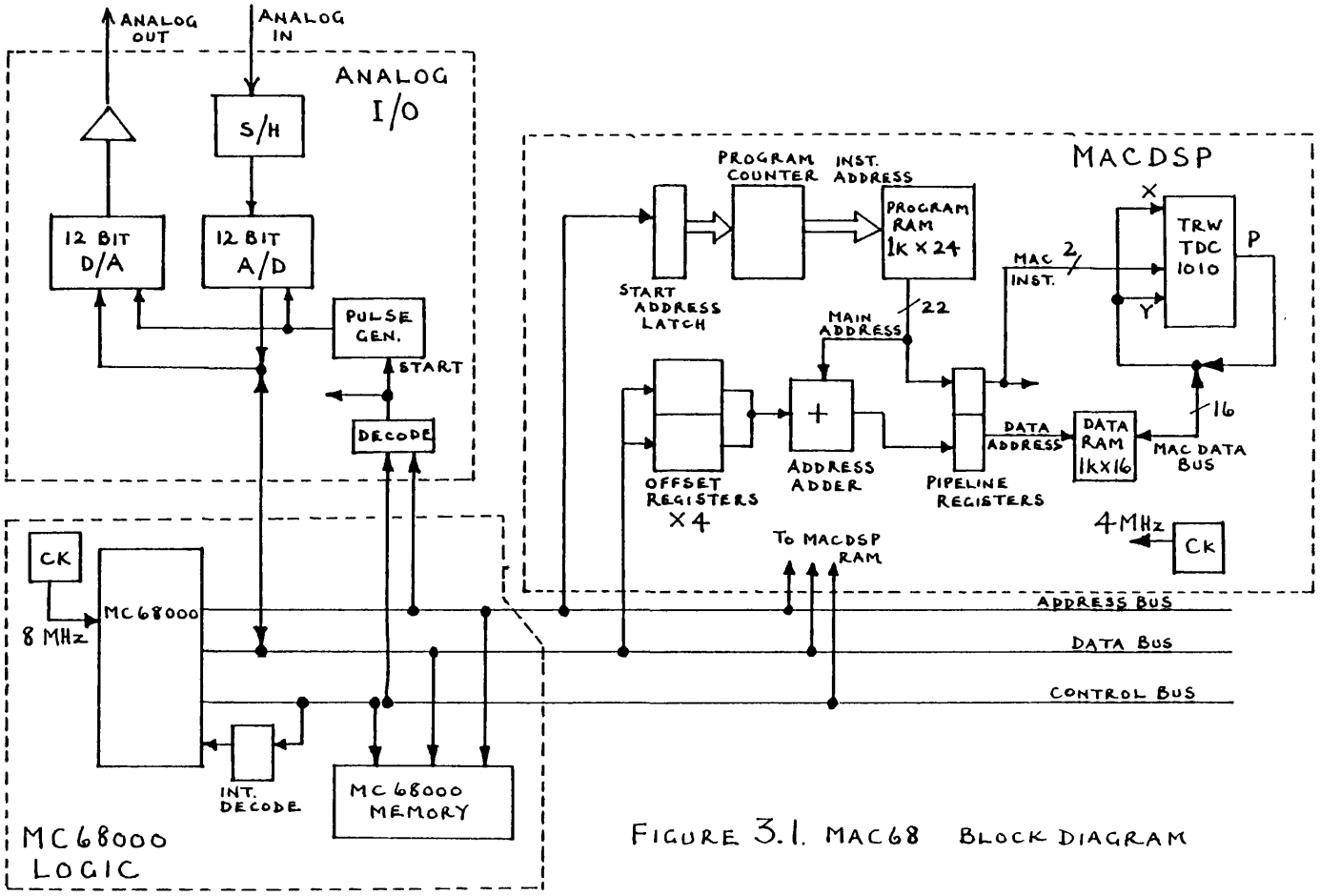


FIGURE 3.1. MAC68 BLOCK DIAGRAM

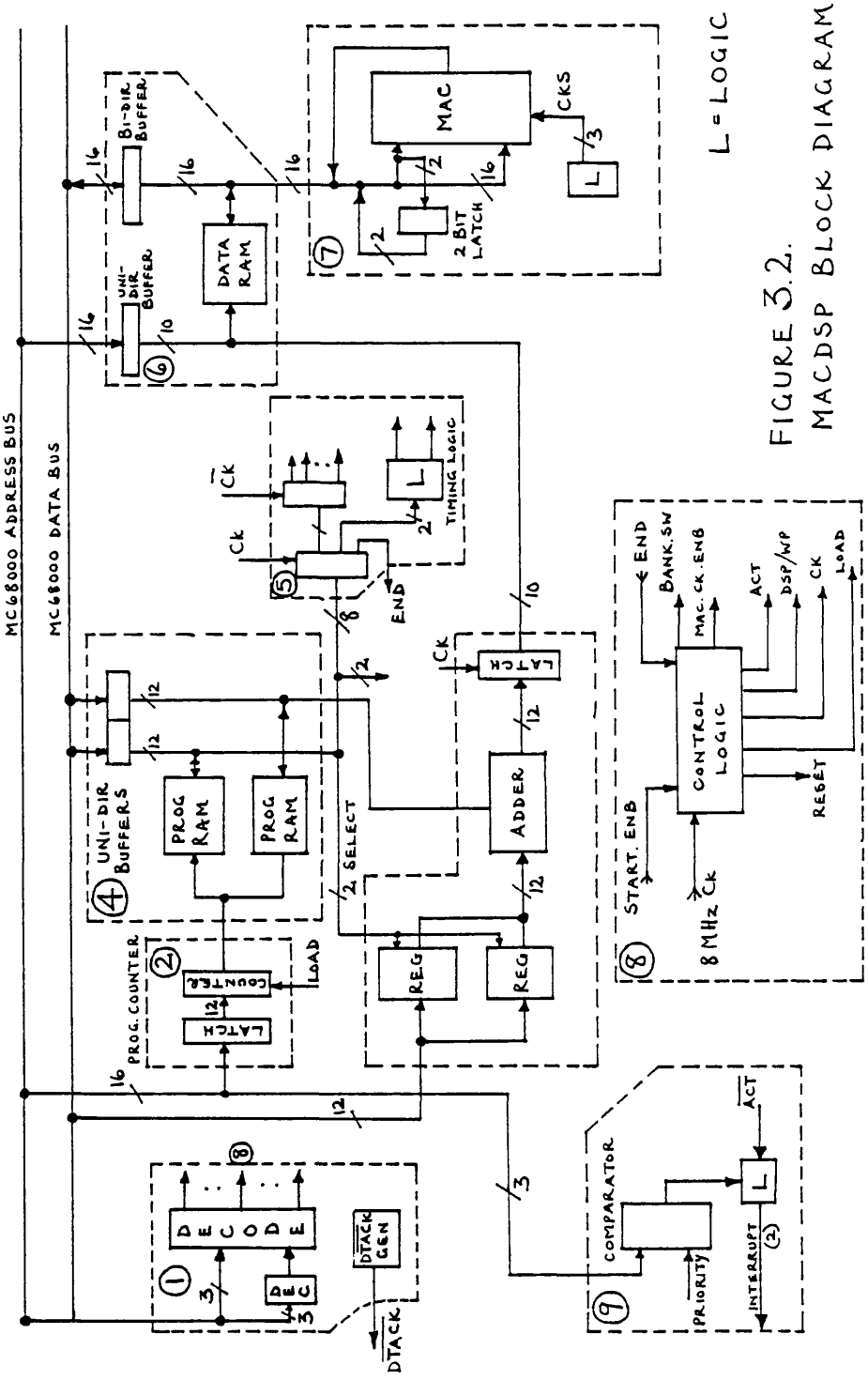


FIGURE 3.2.
MACDSP BLOCK DIAGRAM

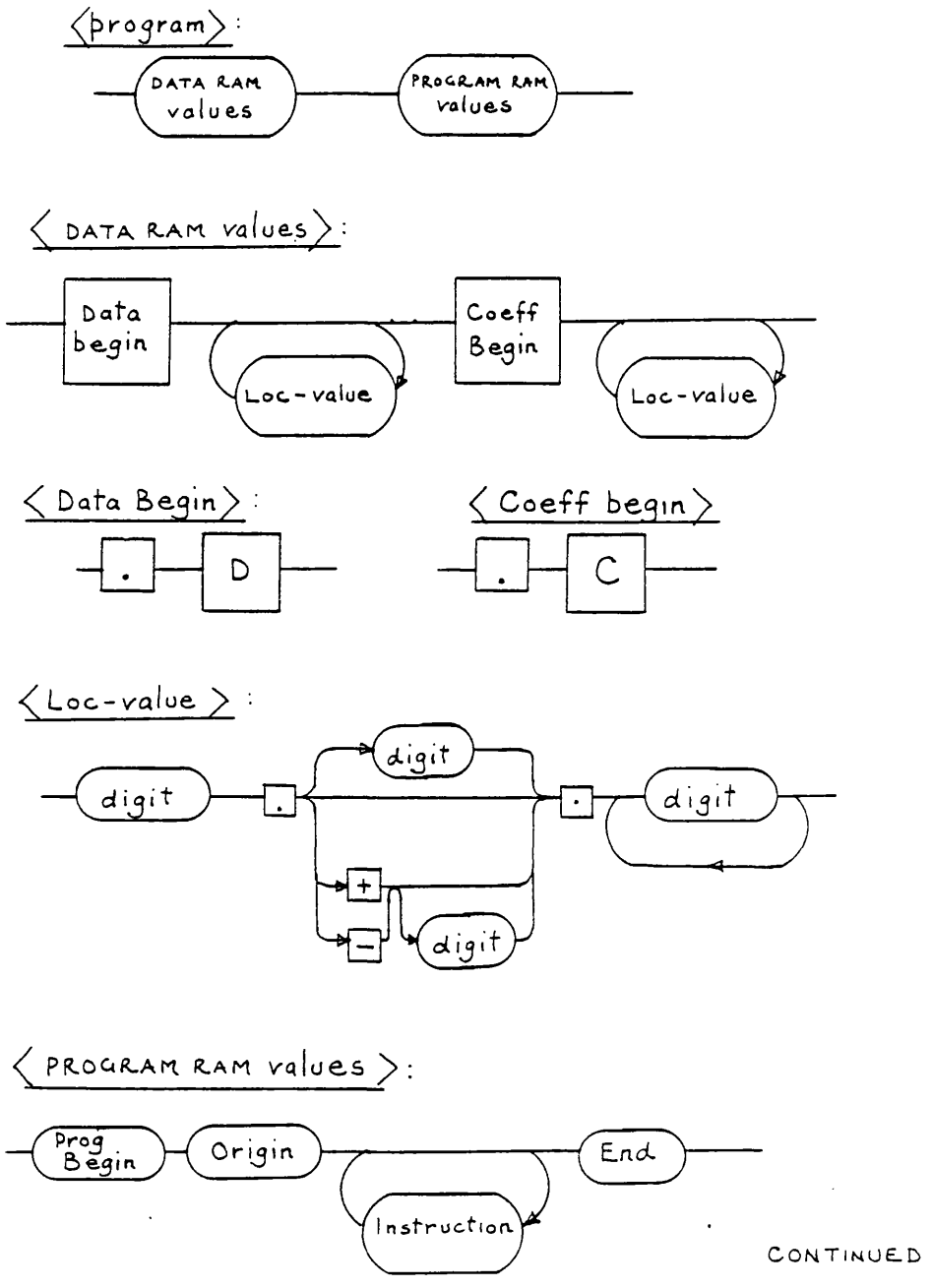
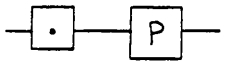
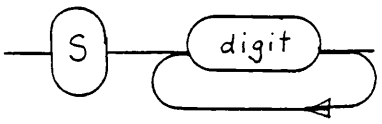


FIGURE 3.3. MACDSP ASSEMBLY LANGUAGE AND SYNTAX

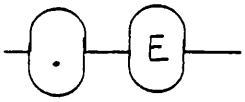
<Prog Begin>:



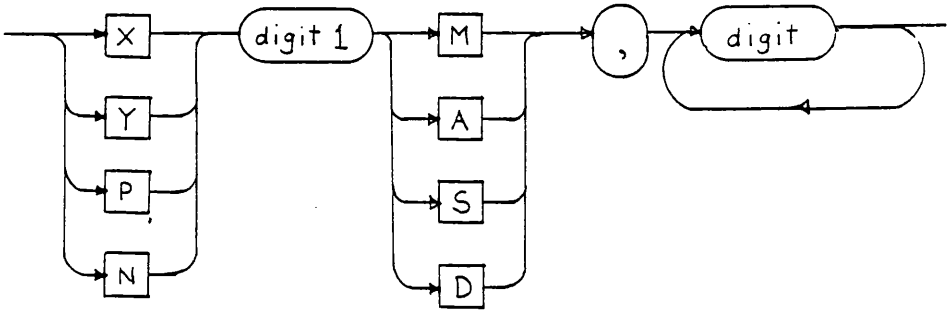
<Origin>:



<End>:



<Instruction>:



<digit>:

$\emptyset|1|2|3|\ldots|9$

<digit 1>:

$1|2|3|4$

FIGURE 3.3. CONTINUED

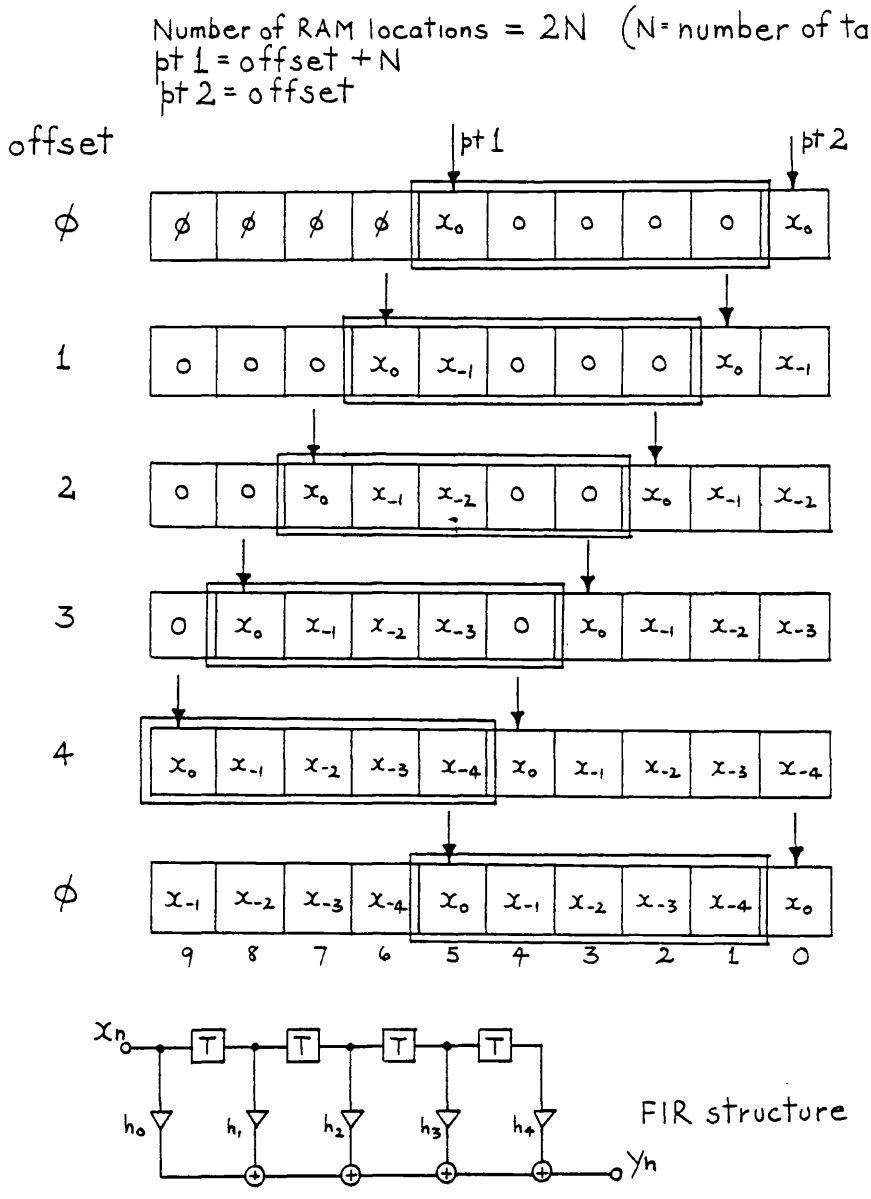


FIGURE 3.4. CIRCULAR BUFFERING TECHNIQUES FOR DIGITAL FILTERS

```
;
;    RZ=RXQ.RC-IXQ.IC
;    IZ=RXQ.IC+IXQ.RC
;
;    RXP'=(RXP+RZ)/2
;    IXP'=(IXP+IZ)/2
;    RXQ'=(RXP-RZ)/2
;    IXQ'=(IXP-IZ)/2
;
;    BUTTERFLY INPUTS:
;        XP=RXP+iIXP=0.5-j0.6
;        XQ=RXQ+iIXQ=0.2+j0.1
;         C=RC+jIC=0.15-j0.25
;
;    BUTTERFLY OUTPUTS:
;        XP'=0.2775-j0.3175
;        XQ'=0.2225-j0.2825
;
;      ( Z=0.055-j0.035 )
;
;
.D
  1,0.5     ;RXP
  2,-0.6    ;IXP
  3,0.2     ;RXQ
  4,0.1     ;IXQ
  10,0.0    ;RZ
  11,0.0    ;IZ
```

Figure 3.5. Radix-2 Complex-butterfly routine for the MACDSP (continued on next page)..

```
.C
  5,-0.5
  6,0.5
  7,0.15    ;RC
  8,-0.25   ;IC
.P
  S0
  X1D,8     ;XREG:=IC
  Y1M,4     ;ACC:=IC*IX0
  X1D,7     ;XREG:=RC
  Y1S,3     ;ACC:=RX0*RC-IC*IX0
  Y1M,4     ;ACC:=RC*IX0
  P1D,10    ;RZ:=ACC
  X1D,8     ;XREG:=IC
  Y1A,3     ;ACC:=RX0*IC+RC*IX0
  X1D,5     ;XREG:=-0.5
  P1D,11    ;IZ:=ACC
  Y1M,1     ;ACC:=-0.5*RXP
  Y1S,10    ;ACC:=(-0.5*RZ)-(-0.5*RXP)
  X1A,6     ;ACC:=0.5*RXP
  P1D,3     ;RX0:=ACC'
  X1A,6     ;ACC:=(0.5*RZ)+(0.5*RXP)
  Y1M,2     ;ACC:=0.5*IXP
  P1D,1     ;RXP:=ACC'
  Y1A,11    ;ACC:=(0.5*IZ)+(0.5*IXP)
  X1A,5     ;ACC:=0.5*IXP
  P1D,2     ;IXP:=ACC'
  X1A,5     ;ACC:=(-0.5*IZ)+(0.5*IXP)
  N1D,0     ;DELAY
  P1D,4     ;IX0:=ACC
.E
```

Figure 3.5 continued.

4. APPLICATION OF THE MAC68 SYSTEM TO THE SUB-BAND CODING OF SPEECH

Speech processing, and in particular speech coding, was chosen as a suitable application area for the MAC68 because of the current research interest in this area, and because the computational capabilities of the MAC68 approximately match those required for real-time processing of speech.

Within the range of speech coding schemes [H-3], Sub-band coding was chosen as the specific application for the MAC68, although other schemes could have been used.

The objective of applying a prototype system, in this case a Digital Signal Computer System, is to validate all system features and to document any weaknesses for later replacement or improvement. The Sub-band Coder implemented did exercise all MAC68 features therefore meeting the objective of the application. Suitable algorithms were firstly developed, then the various implementation tradeoffs attempted, and finally the best Sub-band Coder solution for the MAC68 was implemented and tested.

4.1 Sub-Band Coding of Speech

The Sub-band Coding of speech is intended primarily for digital transmission at data rates in the region of 16Kb/s, which gives an approximate factor of 4 decrease in data rate compared with u-law PCM. This data rate reduction is obtained by exploiting certain properties of speech and hearing mechanisms.

For sub-band coders, each filtered sub-band is coded individually by a waveform coding process using a sampling rate equal to twice the sub-band bandwidth. As sub-bands become narrower the quantizing noise in each band becomes progressively more masked by the speech signal in each band --- an effect due to the ear's critical bands [H-3]. Further, because the power in the higher sub-bands is primarily from 'unvoiced' excitations, the waveform shape in these bands does not have to be specified so accurately, consequently fewer bits can be allocated to these bands. The waveform coding processes most often used in Sub-band Coders are Adaptive Pulse Code Modulation (APCM) and Adaptive Differential Pulse Code Modulation (ADPCM), the latter normally using fixed prediction. The adaptive coding further reduces the number of bits required for quantization (compared with non-adaptive schemes) by matching quantitization step-size to signal variance using a knowledge of the quantizer slot occupied by the previous sample.

Original Sub-band Coder designs incorporated sub-band filters with non-overlapping frequency responses (giving a reverberant quality), or overlapping frequency responses introducing aliasing errors [C-6]. Some of the current Sub-band Coder designs are based on the new technique of Quadrature Mirror Filtering (QMF), whereby sub-band filter responses are allowed to overlap, but where filter properties are such that aliasing errors are minimised. This QMF technique is used for the MAC68 Sub-band Coder implementation.

Consider now the waveform coding for each sub-band. The merit of using fixed prediction for the coding of full-band speech(based on long-term sample-to-sample correlation) is now well established [C-6, J-1]. For sub-band signals this is not the case, as when a large number of sub-bands are used (5 say), the drop in power density across any one band is relatively small. As sub-bands are in effect, translated to low-pass and sampled at their Nyquist rate, they appear essentially as flat spectrum signals at the low sub-band sampling rates, and therefore have essentially no sample-to-sample correlation [C-6]. Therefore Adaptive Pulse Code Modulation was chosen, and the assumption was made that a sufficient number of sub-bands would be used, such that fixed prediction is not necessary.

To summarise, the proposed Sub-band Coder scheme uses, Quadrature Mirror Filtering as the sub-band filtering method, and Adaptive Pulse Code Modulation as the waveform coding process. A full-duplex Sub-band Coder was developed, such that the A/D values feed into the transmit input, the Transmit ouput feeds into the Receive input (internal to the processors), and the Receive output feeds into the D/A. Note that the implementation models this configuration and that no physical Transmit output is obtained.

4.1.1 Quadrature Mirror Filters

Appendix VIII details the development of the computation sturcture for 2-band Quadrature Mirror Filtering. The band spliting process can be extended indefinitely to produce a tree structure or tree-QMF. Consider for example a uniform 8-band tree-QMF structure where each stage of the tree uses an identical 2-band QMF (or QMF building block). Each of the sub-band frequency responses are then given by (Figure 4.1):-

$$H1(w) = H(w).H(2w).H(4w) \qquad (4.1)$$

$$H2(w) = H(w).H(2w).H(4w+\pi) \qquad (4.2)$$

$$H3(w) = H(w).H(2w+\pi).H(4w) \qquad (4.3)$$

$$H4(w) = H(w).H(2w+\pi).H(4w+\pi) \qquad (4.4)$$

$$H5(w) = H(w+\pi).H(2w).H(4w) \qquad (4.5)$$

$$H6(w) = H(w+\pi).H(2w).H(4w+\pi) \qquad (4.6)$$

$$H7(w) = H(w+\pi).H(2w+\pi).H(4w) \qquad (4.7)$$

$$H8(w) = H(w+\pi).H(2w+\pi).H(4w+\pi) \qquad (4.8)$$

Where H(w) is the low-pass frequency response for the prototype 2-band QMF (and $H(w+\pi)$ the corresponding high-pass frequency response) and w represents normalised frequency i.e. wTs. Hn(w) therefore represents the frequency response for the nth sub-band. Each product term in the equations corresponds to one of the stages in the tree structure, consider for example H5(w):

$H5(w) = H(w+\pi).H(2w).H(4w).$

$H(w+\pi)$ - High-pass path through stage 1.

H(2w) - Low-pass path through stage 2. One level of decimation gives 2w.

H(4w) - Low-pass path through stage 3. Two levels of decimation gives 4w.

From these equations it is clear that individual sub-band frequency responses will have unequal slopes, even though the QMF principle still holds. Filter slopes can be made more equal, by using 2-band QMFs of different orders for each of the tree stages, so that the transition bandwidths are kept approximately equal. For example the filter order could be halved at each stage [G-6].

The tree structure given in Figure 4.1., has an equal amount of decomposition for each branch. When the decomposition is unequal, such that some branches extend by a greater number of stages than other branches, aliasing can only be avoided by replacing the missing band-split structures with equivalent non band-spliting structures [G-6]. In general therefore, it is desirable to minimise the number of non-uniform band splits by as much as possible.

Two popular QMF methods, based on the tree-QMF computation structure, which are used for more than 2-bands are:-

(1) Tree-QMF Implementation.

This is where the tree-QMF computation structure is implemented directly.

(2) Parallel-QMF Implementation.

Parallel-QMF structures are derived from the tree-QMF structure by using the Impulse Response for each of the tree-QMF paths. Impulse Responses are reduced in length as far as possible without the overall error increasing significantly [G-6], therefore reducing the amount of computation.

The computational requirements for the parallel-QMF implementation are generally less than those for the tree-QMF implementation, although coefficient design for the former is more complicated. For the MAC68, the tree-QMF implementation method was chosen, as coefficients for tree-QMF building blocks are readily available, so that complete tree-QMF structures can be built up without further coefficient design. Further, at the time of the application of the MAC68, sufficient data was not available on the Parallel-QMF implementation method. NOTE: Parallel QMF structures require a greater number of coefficients than tree-QMF structures, this could restrict their implementation on the MAC68.

Coefficients for all QMF implementations on the MAC68 were taken from [J-2].

4.1.2 Adaptive Pulse Code Modulation (APCM).

Figure 4.2 shows the general form of the APCM algorithm. For the robust adaption scheme [G-5], the new step-size S(n+1) is chosen by:-

$$S(n+1) = (S(n))^{Y} .M(|I(n)|) \quad (4.9)$$

$$SMIN < S(n+1) < SMAX$$

where I(n) is the previous code word, and Y is a constant, typically 0.98. With Y < 1 error effects decay exponentially with time, so that the quantizer is robust in the presence of transmission errors. The function M(.) of the code word I(n) is such that if I(n) is an upper magnitude level, a value greater than one is used. Conversely for lower magnitude levels a value M(.) less than one is used (see Table 4.1). In this way the quantizer adapts to the range of the input signal.

Reference [J-1] gives the general form of optimal multiplier functions for speech quantization. From these functions it can be seen that step-size increases are more rapid than step-size decreases. This is because granular errors are less 'harmful' than overload errors [J-1]. Multiplier values used are given in Table 4.1, these are taken from reference [C-6].

The ratio of SMAX to SMIN in APCM, determines the dynamic range of the coder, and the absolute values of SMAX and SMIN determine the centre of this dynamic range. The ratio used in [C-6] is:

$$\frac{SMAX}{SMIN} = 128 \qquad (4.10)$$

which gives a useful dynamic range of about 40dB. The absolute values of SMAX and SMIN are determined from the long term spectrum of speech. Since upper sub-bands have lower power densities than lower sub-bands, they must use smaller values of SMAX and SMIN.

For the MAC68 implementation of APCM, values of SMIN (and therefore also SMAX) for each sub-band were determined by firstly assuming a flat power density for each sub-band and obtaining the approximate mean value, Pi (for the ith sub-band). The long-term variance for the ith sub-band is then proportional to PiFi, where Fi is the sub-band bandwidth, therefore the values of SMIN can be obtained using the relation [C-6]:

$$\frac{SMIN\ (band\ i)}{SMIN\ (band\ j)} \simeq \sqrt{(PiFi/PjFj)} \qquad (4.11)$$

See section 4.3.1 for the design of the final SMAX and SMIN parameters.

The following minimisations were applied to the above basic APCM algorithms to ease the MAC68 implementations:

(1) Use of Y=1.

Using Y=1 makes the APCM implementation non-robust, which in these circumstances was acceptable in that no actual data transmission was required. In using Y=1 the implementation is simplified, (although the computation time not necessarily reduced) as the need for the exponent part of the algorithm is eliminated, so that a straight-forward division can be used to replace the look-up tables that are normally used [C-10].

(2) Reduced Computation.

Common sections of the APCM encoder and decoder for each band were combined to produce the reduced computation structure shown in Figure 4.3. This relegates the APCM implementation to a real-time simulation, although this is sufficient to give the full effects of adaptive coding.

NOTE:

A MC68000 implementation of this final APCM computation structure was tested using computer generated samples. This was done to ensure that the program logic was correct prior to the incorporation of the algorithm into a complete Sub-band Coder.

4.2 Implementation Tradeoffs on the MAC68 System

Given the tree-QMF and APCM computation structures presented in Section 4.1, the various tradeoffs available in implementing these structures on the MAC68 can be discussed. The ultimate objective of these discussions is to find the best possible Sub-band Coder for the available hardware, suitable figures of merit being the number of bands used, and the number of taps used in tree-QMF building block(s).

Consider firstly the two most fundamental methods for implementing tree-QMFs on the MAC68:

(1) A single expanded (non-repeated) MACDSP sequence for the complete tree-QMF.

(2) The repeated execution of MACDSP sequences which represent the tree-QMF building blocks.

Initial studies showed that because of MACDSP program memory limitations, large tree-QMF structures could not be implemented using method (1). For example, one of the largest structures possible with (1) would be:

4 bands: Stage 1 --- 64 taps for each building block.
Stage 2 --- 32 taps for each building block.

using circular-buffering (see Chapter 3) and a uniform band split.

Estimates showed that larger structures could be implemented for method (2), consequently this method was adopted. Method (2) is now further sub-divided into two basic sub-methods; self-start and non self-start. In using non self-start it was felt that the full potential of the MACDSP would be under-utilised, and that it would not allow the full evaluation of all MAC68 features. Consequently, of the tree-QMF implementation methods studied, the repeated execution of MACDSP sequences utilising the self-start mechanism seems most suitable.

As discussed in chapter 3, there are not insubstantial overheads associated with each interrupt. These overheads can be minimised by combining the Transmit and Receive MACDSP sequences for the tree-QMF building block(s) into a single MACDSP sequence, therefore reducing the number of MACDSP interrupts by approximately a factor of 2. The immediate consequences of this, is that buffering is now required between the three sections, Transmit QMF, APCM and Receive QMF. Further, because there is an insufficient number of offset registers to allow the outputs of each QMF sequence execution to be loaded into the relevant input locations of subsequent QMF sequence executions, an additional MACDSP sequence is required, a Data move sequence. This Data move sequence effectively implements an 8-sample period delay between each stage in the tree-QMFs.

The Data move sequence is used to ensure that the path delay for all samples along the tree branches, is identical. In general, the use of buffering in the above ways increases the overall input-output delay for the complete Sub-band Coder, although these delays would be acceptable in most full-duplex communication systems [B-11]. It should be pointed out that the delays in a Sub-band Coder are quite conveniently implemented because of the straight-forward cascading of QMF and APCM blocks, for other applications this might not be the case, and buffer movement overheads or overall delays could be excessive.

A further simplification is now added to the above, this is to use a single tree-QMF building block for all parts of the QMFs. This means that the equal transition band-width rule mentioned in Section 4.1 is not used, and that equations 4.1-4.8 are now applicable. This simplification is used so that the timing of concurrent processes is more straight-forward, a necessity introduced by the timing problem discussed in chapter 3.

The final form of the tree-QMFs which minimises the number of MACDSP interrupts, and efficiently utilises all MAC68 features, can now be described:-

(1) A single tree-QMF building block (MACDSP sequence) consisting of the Transmit and Receive sections of a 2-band QMF. We shall call this MACDSP sequence the TQMF-BB (tree-QMF building block). See Figure VIII.1 for the relevant computation structure.

(2) Repeated execution of this tree-QMF building block using self-start. The input values for each TQMF-BB are taken from an input-output table held in MACDSP data RAM, and output values for each TQMF-BB are stored in the same table. An offset register is loaded up for each TQMF-BB to point to the relevant position in this table.

(3) Execution of a Data Move sequence by the MACDSP once tree-QMF computation has taken place, implementing the inter-stage movement of data for the tree-QMFs.

Having got to this stage, some basic issues relating to overall Sub-band Coder implementation are still left unresolved. These issues primarily concern the use of circular buffering for the QMFs and how this affects the implementation of APCM on the MAC68. Figures 4.4 and 4.5 give the fraction of total available computation time used for MACDSP sequence executions (not including MC68000 processing times), for the above chosen tree-QMF form, with and without the use of circular-buffering. Values are plotted for variation of:-

- Number of sub-bands used.
- Number of taps used in the TQMF-BB.

These graphs are for implementations which meet MACDSP data and program memory limitations. Note that circular-buffer implementations are data memory limited, and non circular-buffer implementations are computation time limited. This is because the TQMF-BB for the circular-buffer case uses half the number of instructions of the TQMF-BB for the non circular-buffer case, but, at the same time uses twice the number of data memory locations.

NOTE:
It is possible to implement the circular-buffer method on the MAC68 without doubling MACDSP data memory requirements, by using three different start addresses for a single QMF sequence. But this then requires three MACDSP interrupts with all the associated timing problems. See chapter 5 for details of this method applied to an enhanced MAC68 system.

The issues involved are fundamental to the MAC68 concept, and a more detailed comparison of the implementation of the two methods, circular and non circular-buffering, does in fact highlight the different ways concurrent processes can be used in the MAC68. This comparison can now be carried out.

Trial implementation of all the tree-QMF methodologies have been successfully carried out.

4.2.1 Circular-Buffering

The first observation to make for this method concerns the complexity involved in calculating offet addresses. The implementation of circular-buffering for a multi-stage tree-QMF is complicated by the fact that the circular buffers for each tree stage are 'circulating' at different rates. As for example for an 8band/3stage tree-QMF structure, the first stage TQMF-BB is executed 4 times, the second stage TQMF-BB 2 times etc., so that the same circular-buffer parameters cannot be used for all TQMF-BBs executed. A result of this is that TQMF-BBs should be executed sequential with the APCM routines, as opposed to allowing APCM to be executed as a background task which is active when the MC68000 is not servicing the MACDSP or the I/O.

Concurrent processes would be set up such that the main MACDSP driving routine would be on priority level 1, and this routine would loop (once the MACDSP had been started) until a flag is set by the MACDSP ISR indicating that the MACDSP had terminated. This process would execute the tree-QMF and APCM routines concurrently with I/O, which would be serviced by the I/O ISR on the highest priority level. Overall process synchronisation would be provided by an I/O ISR every complete program period, this I/O ISR would also carry out the buffer movement between the various Sub-band Coder components. Note that the MACDSP çannot be active while this inter-processor buffer movement is in progress.

APCM execution times could be minimised by grouping the arithmetic from the APCM for each of the bands used, and executing it as a single MACDSP sequence.

Consider now the size of tree-QMF that could be implemented with this method. Firstly, to maintain self-start the TQMF-BB would have to use at least 48 taps (double this for Transmit and Receive sections) i.e. the MACDSP execution time must balance the offset address calculation, interrupt and I/O ISR times to avoid timing problems. Using Figure 4.4 it can be seen that the best choice is then 5 bands/48 taps, using 52% of the total available execution time. The APCM and Buffer movement routines can then easily execute in the remaining 48% of total computation time. MACDSP data memory limitations prevent 6 or more bands being used with a 48 tap TQMF-BB.

The circular buffer method would utilise all four of the offset registers, using one for containing zero, another for the input - output table and the remaining two for data offsets (i.e. containing circular-buffer addresses).

Figure 4.6 illustrates the execution of processes other than the I/O ISR process.

4.2.2 Non-Circular Buffering

A completely different approach is adopted here, whereby the TQMF-BB uses MACDSP instructions to implement the sample delay operations i.e. physical and not logical data moves. The MACDSP servicing required to execute complete tree-QMFs with this method is much simpler than for the circular-buffer case, consequently the spare MC68000 execution time created, can be used for the APCM routines i.e. the APCM can be executed concurrently with the tree-QMF routines, with all APCM processing being carried out by the MC68000.

Concurrent processes would be arranged so that the priority level 1 routine carries out the APCM processing, while the MACDSP ISR on priority level 2 would be used for servicing the MACDSP. This method of using concurrent processes is clearly quite different to that proposed for the circular buffer case, and at this stage there is no obvious preference.

From Figure 4.5 the most likely candidate for implementation would be the 7bands/32tap case which uses 82% of the total available execution time. The remaining 18% would be allocated for the I/O ISR which synchronises the concurrent activities and carries out buffer movement (during which period the MACDSP cannot be active). It should be pointed out that the timing problem is worse here than for the circular-buffer case, because of the use of slow MC68000

instructions such as DIV in the APCM. The TQMF-BB execution time must therefore be even greater if this problem is to be avoided i.e. a 32 tap TQMF-BB for the non-circular case takes one and a half times longer to execute than a 48 tap TQMF-BB for circular buffering.

Because the APCM execution time on the MC68000 is of the order of 90usec, to repeat this 7 times (for the 7 bands) would take approximately 630usec i.e. 63% of the total available execution time. As 20 interrupts each of 8usec are required, plus the MACDSP ISR executions, it would be safer to repeat the APCM routine only 6 times (using the same tree-QMF structure) and eliminate processing of speech in the 3-4KHz band, thus saving a valuable 90usec of MC68000 computation time.

Figure 4.7 illustrates the interaction of all processes except the I/O ISR process.

4.3 A 16Kb/s 6-Band Sub-Band Coder

The non circular-buffer method of section 4.2.2 was chosen for the final implementation strictly on the basis of the greater number of sub-bands possible (there was not sufficient time to implement both methods). A data rate of 16Kb/s was used as opposed to a lower data rate, as this seemed most suitable for good quality speech considering the sophistication of the Sub-band Coder [C-9,H-3]. With more bands/taps lower data rates could be considered.

The complete Sub-band Coder configuration is illustrated in Figure 4.8. Table 4.2 lists the coefficients used for the 32tap FIR QMF structure (taken from 32C of [J-2]).

NOTE:
Chapter 5 discusses hardware and software improvements to the MAC68 system, whereby a method more similar to that of section 4.2.1 would be preferable.

4.3.1 APCM Parameters

Now that the final Sub-band Coder computation structure is decided on, there remains the task of determining APCM parameters for this structure.

The bit allocation for the sub-bands is similar to that given in [C-9], and is:-

BAND1 (0.0-0.5KHz): 4 bits.
BAND2 (0.5-1.0KHz): 3 bits.
BAND3 (1.0-1.5KHz): 3 bits.
BAND4 (1.5-2.0KHz): 2 bits.
BAND5 (2.0-2.5KHz): 2 bits.
BAND6 (2.5-3.0KHz): 2 bits.

Minimum and maximum step sizes for each band were derived using, equatins 4.10 and 4.11, the long term speech spectrum given in reference [C-6], and the assumption that SMAX for band1 is 0.125. This value for SMAX (band 1) is obtained by allowing a maximum input value of 1.0 for band1 APCM, so that overflow is prevented for this band and for the other bands. The initial value of step-size was arbitrarily set to 0.05 for all bands.

Table 4.3 gives the scale factors derived from the speech spectra in reference [C-6]. These factors are used instead of Pi and Pj in equation 4.11 (as all sub-bands are equal).

4.3.2 Control of Concurrent Processes

To minimise the overheads involved in switching between concurrent processes in the MC68000, contents of general-purpose registers are not stored/restored for interrupts. This is made possible by maintaining fixed address constants in the MC68000 address registers, and by sharing the MC68000 data registers between the different concurrent processes. Real-time execution is initialised by jumping to an initialisation routine which sets up the address constants, clears the I/O loop count and starts the I/O counter.

The following sections summarise details of the functions carried out by the Interrupt Service Routines and by the main routine. Figures 4.9 and 4.10 give flow diagrams for the initialisation and 'normal' execution of the concurrent processes for the Sub-band Coder.

4.3.2.1 Input-Output Interrupt Service Routine Functions

On every I/O interrupt, a value is fetched from the A/D and converted from bipolar to 2's complement format and then loaded into the I/O input buffer. Also, a 2's complement value is taken from the I/O output buffer converted to bipolar format and loaded into the D/A.

On every eighth I/O interrupt the I/O ISR initiates a complete Sub-band Coder program cycle. Other than handling Analog I/O this initialisation involves:-

(1) Moving data from the I/O input buffer into the TQMF-BB input-output table, and from the TQMF-BB input-output table to the I/O output buffer.

(2) Moving data from the TQMF-BB input-output table to the APCM input buffer, and from the APCM output buffer to the TQMF-BB input-output table.

(3) Setting a flag to enable APCM execution.

(4) Initiating the first MACDSP sequence executions.

4.3.2.2 MACDSP Interrupt Service Routine

The MACDSP ISR is used 12 times throughout the entire Sub-band Coder program sequence, this corresponds to:-

Stage 1 -- 4 MACDSP ISRs

Stage 2 -- 4 MACDSP ISRs

Stage 3 -- 3 MACDSP ISRs

Data Move -- 1 MACDSP ISR

The Data Move sequence is the last MACDSP sequence to be executed, and requires a new MACDSP start address to be loaded into the MACDSP by the eleventh MACDSP ISR (the twelfth MACDSP ISR executes RTE only). Note that because the Data Move sequence is the last to be executed, the self-start timing problems do not apply to it.

The TQMF-BB input-output table is a continuous area of MACDSP data RAM. As the TQMF-BB is executed 11 times, the input-output table consists of 88 locations i.e. 4 inputs and outputs for the Transmit and Receive parts of the TQMF-BB. Three of the offset registers are used, one for holding zero, one for the input-output table and the third for pointing to data areas. Delay variable data is organised so that only the one data offset is required for pointing to delay variable data areas. All offset addresses were pre-calculated to save computation time.

The following fragments from the MACDSP program for the TQMF-BB illustrate how values are moved from the input-output table into the delay variable area for the transmit QMFs:-

```
;     OFFSET REGISTER 1=0
;     OFFSET REGISTER 2= DATA OFFSET
;     OFFSET REGISTER 3= INPUT-OUTPUT TABLE OFFSET
.D
      0,0.0  ;  DATA  LOCATION  CORRESPONDING  TO  TAP1  OF  THE
             ;  UPPER TX.
     16,0.0  ;  DATA  LOCATION  CORRESPONDING  TO  TAP1  OF  THE
             ;  LOWER TX.
    900,0.0 ; LOCATION IN IN-OUT TABLE.
    901,0.0 ; LOCATION IN IN-OUT TABLE.
      --------------------
.C
   1016,1.0 ; CONSTANT +1.0
      --------------------
.P
      SO
    X3D,900  ;  LOAD  INPUT  VALUE  FOR  UPPER  BRANCH  INTO  X.
    Y1M,1016 ; PUT THIS VALUE INTO THE ACCUMULATOR.
    X3M,901  ;  LOAD  INPUT  VALUE  FOR  LOWER  BRANCH  INTO  X.
    P2D,0 ; STORE (900) TO UPPER BRANCH.
    P2D,16 ; STORE (901) TO LOWER BRANCH.
      -------------------
.E
```

4.3.2.3 APCM Routines

Three separate APCM routines are used, corresponding to each of the different numbers of bits quantization used. Using different routines minimises the need for indexing, therefore reducing computation times further.

Once the APCM routines have been executed for all 6 bands, processing on this priority level cannot continue until the I/O ISR has set the relevant enabling flag.

4.3.3 Results

With all the above Sub-band Coder components executing, experimental results were obtained. These took the form of:-

(1) Plots of sub-band frequency responses. Figure 4.11 illustrates the experimental configuration, as well as the points in the Sub-band Coder diagram from which plots are taken. Figures 4.12 and 4.13 compare experimental and theoretical results for Band 3 (1-1.5kHz) and Band 5 (2-2.5kHz). The transition band for Band 5 does not exceed 500Hz, therefore because this is the widest transition band for all the sub-bands (being equal to that for band 4), the amount of aliasing caused by sub-band overlap is minimal. The noise level was found to be approximately 45dB down on the pass-band level. As this figure is worse than that anticipated for quantization effects alone, other noise effects such as digital crosstalk and ground noise must have increased the overall noise significantly.

The comparison between theoretical and experimental values is sufficiently close to verify the band-spliting function.

(2) A recording of 16Kb/s sub-band coded speech was made, and compared with 16Kb/s Continuous Variable Slope Delta modulated (CVSD) speech. The quality of the sub-band coded speech appeared to be good, and compared favourably with the CVSD speech, although no formal subjective tests were carried out.

Female speech appeared less intelligible than male speech for the sub-band coded speech, this is possibly because of the lack of coding in the 3-4kHz bands. This could be partially overcome by the transmission of sign bit information only for these upper bands.

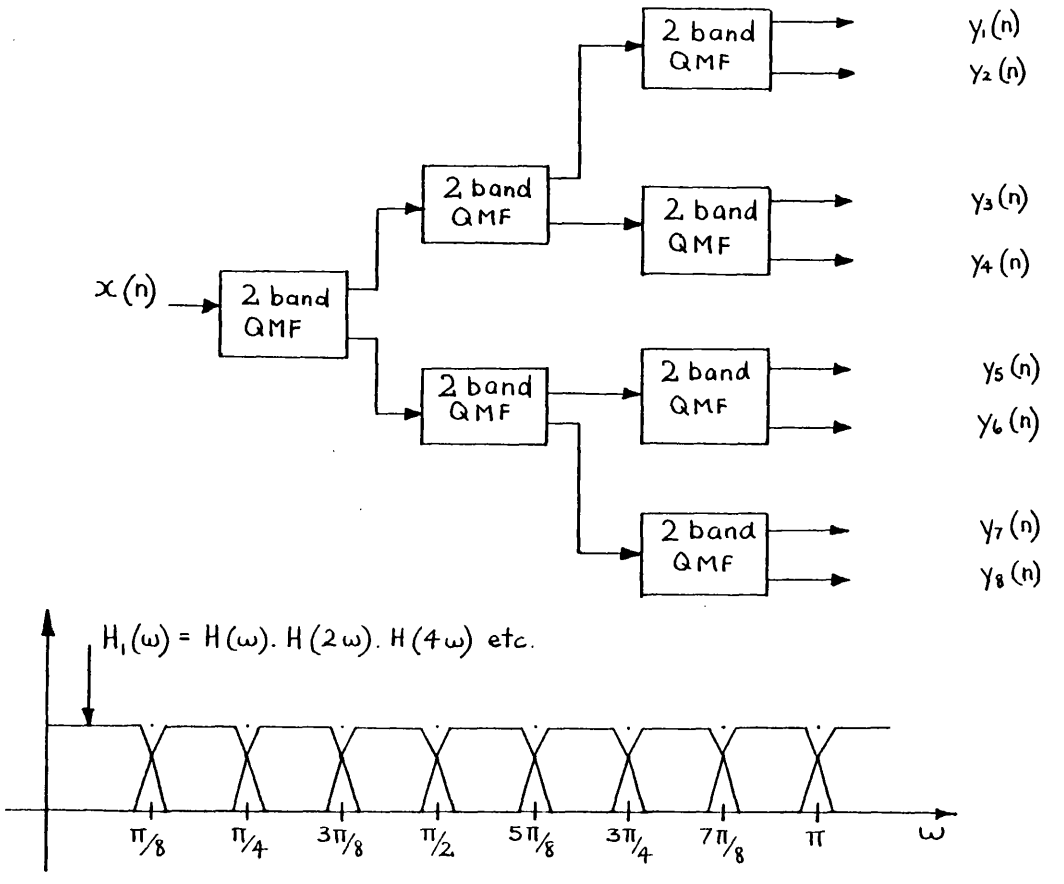


FIGURE 4.1
8-BAND/UNIFORM
TREE-QMF STRUCTURE

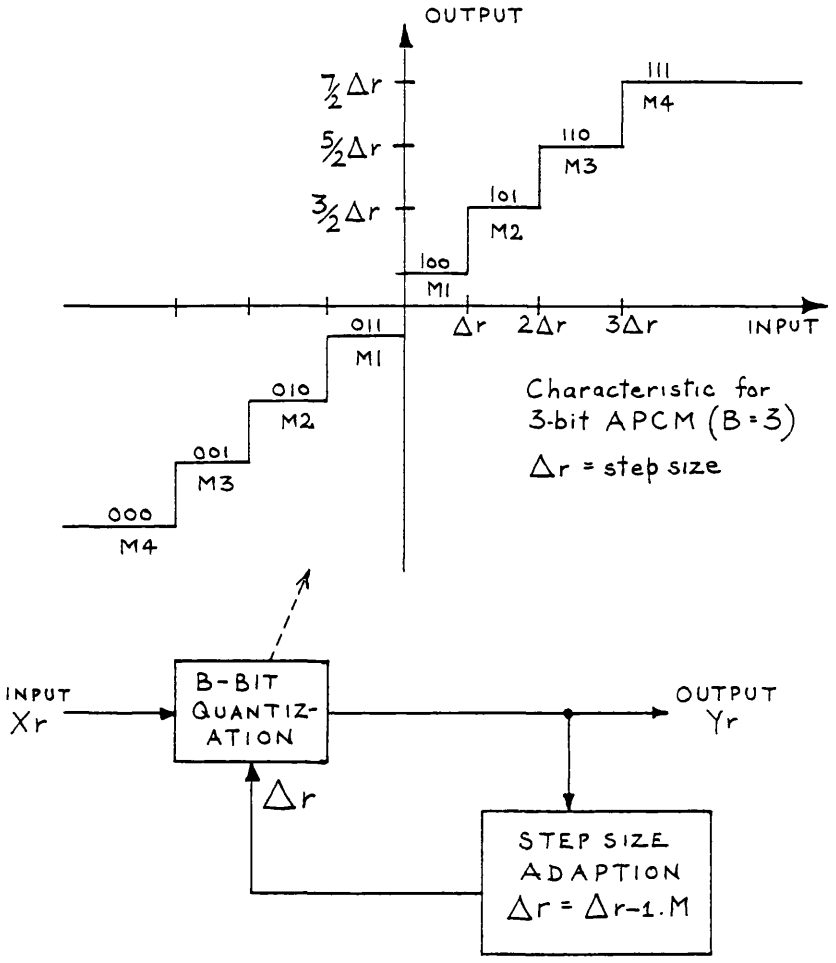
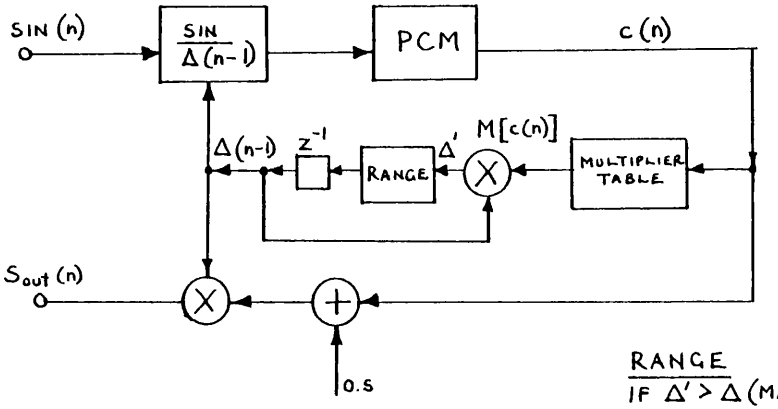


FIGURE 4.2. GENERAL FORM FOR ADAPTIVE PULSE CODE MODULATION (APCM).



SIN - Input value
S_{out} - Output value
C - Code value
Δ - Step size
M - Multiplier value

RANGE
IF $\Delta' > \Delta(MAX)$ THEN $\Delta := \Delta(MAX)$
ELSE
IF $\Delta' < \Delta(MIN)$ THEN $\Delta := \Delta(MIN)$

FIGURE 4.3. APCM STRUCTURE IMPLEMENTED.

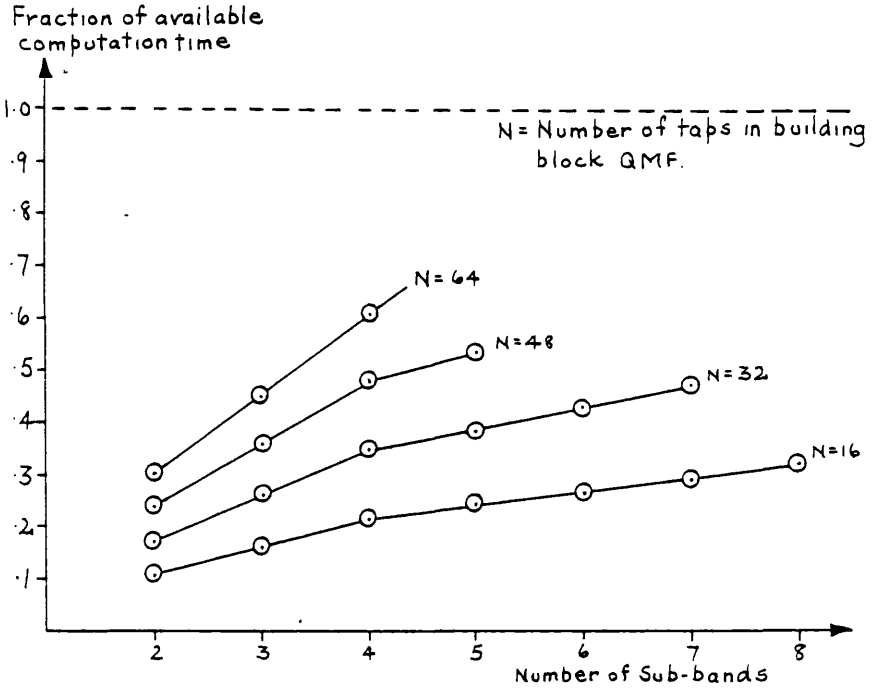


FIGURE 4.4. TREE-QMF IMPLEMENTATION FIGURES FOR CIRCULAR-BUFFERING

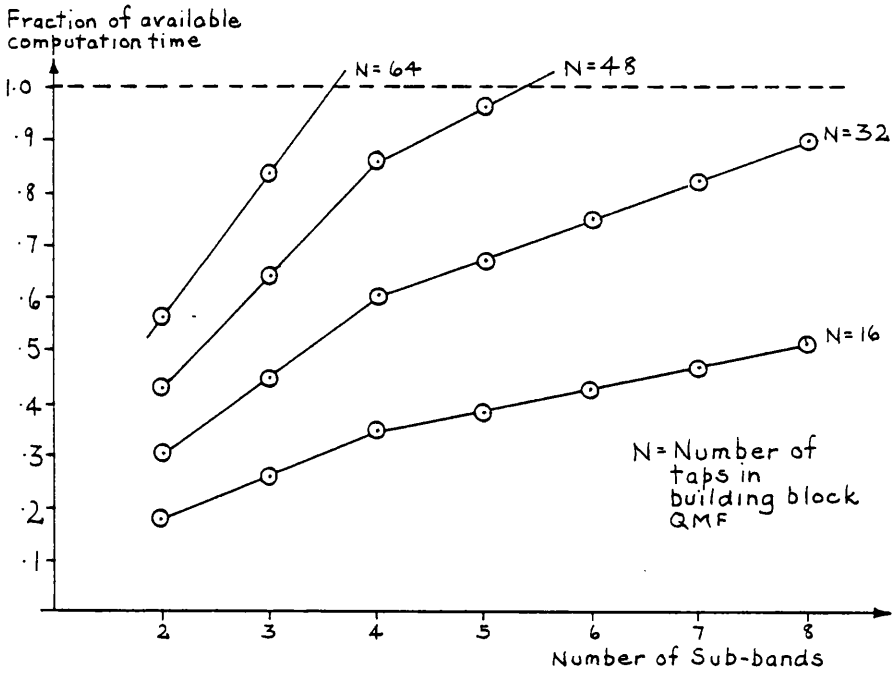


FIGURE 4.5. TREE-QMF IMPLEMENTATION FIGURES FOR NON-CIRCULAR BUFFERING.

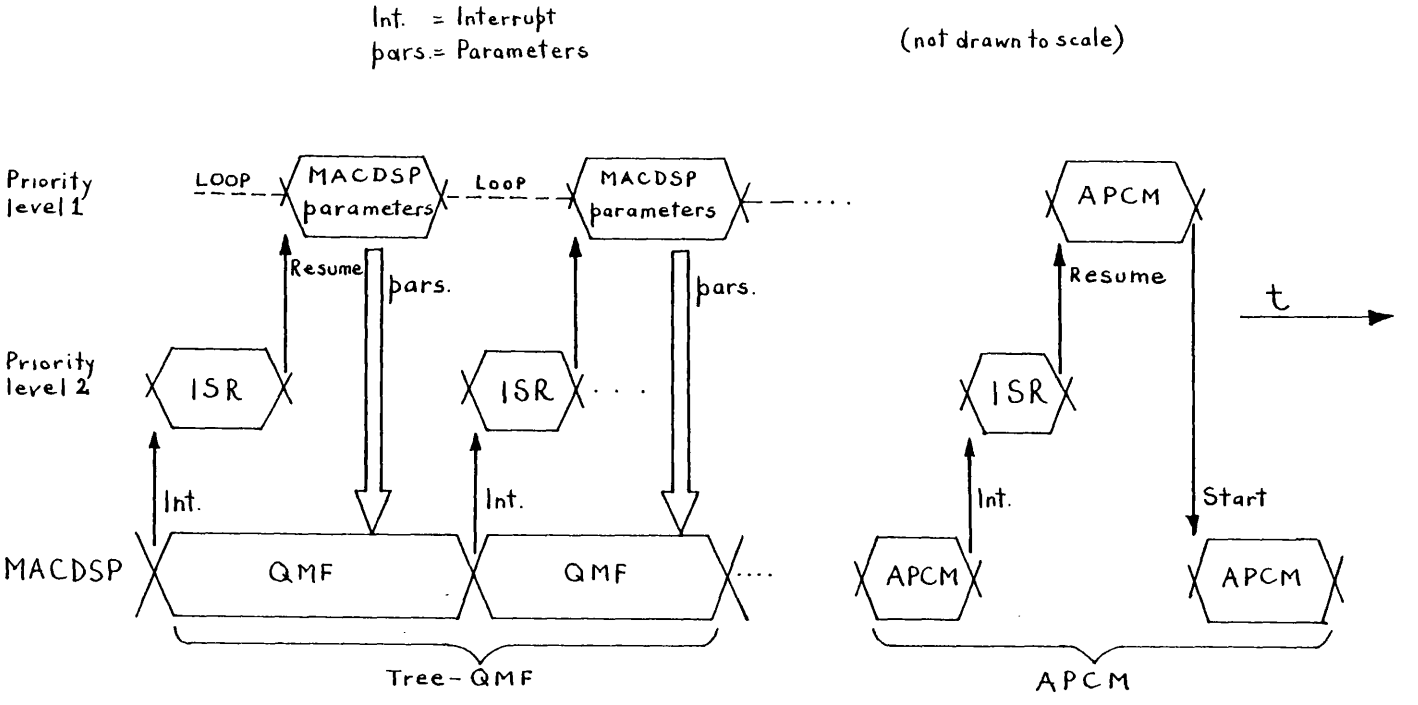


FIGURE 4.6. EXECUTION OF PROCESSES FOR THE CIRCULAR-BUFFER CASE

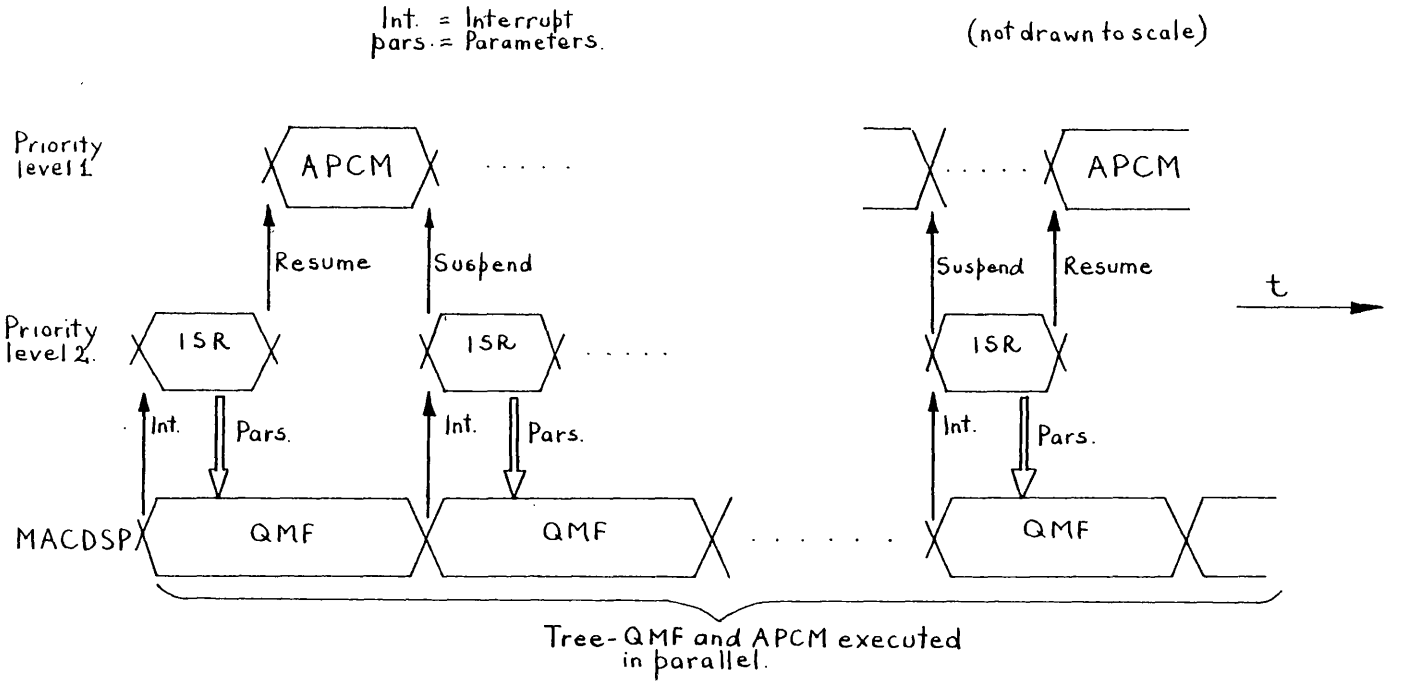


FIGURE 4.7 EXECUTION OF PROCESSES FOR THE NON CIRCULAR-BUFFER CASE

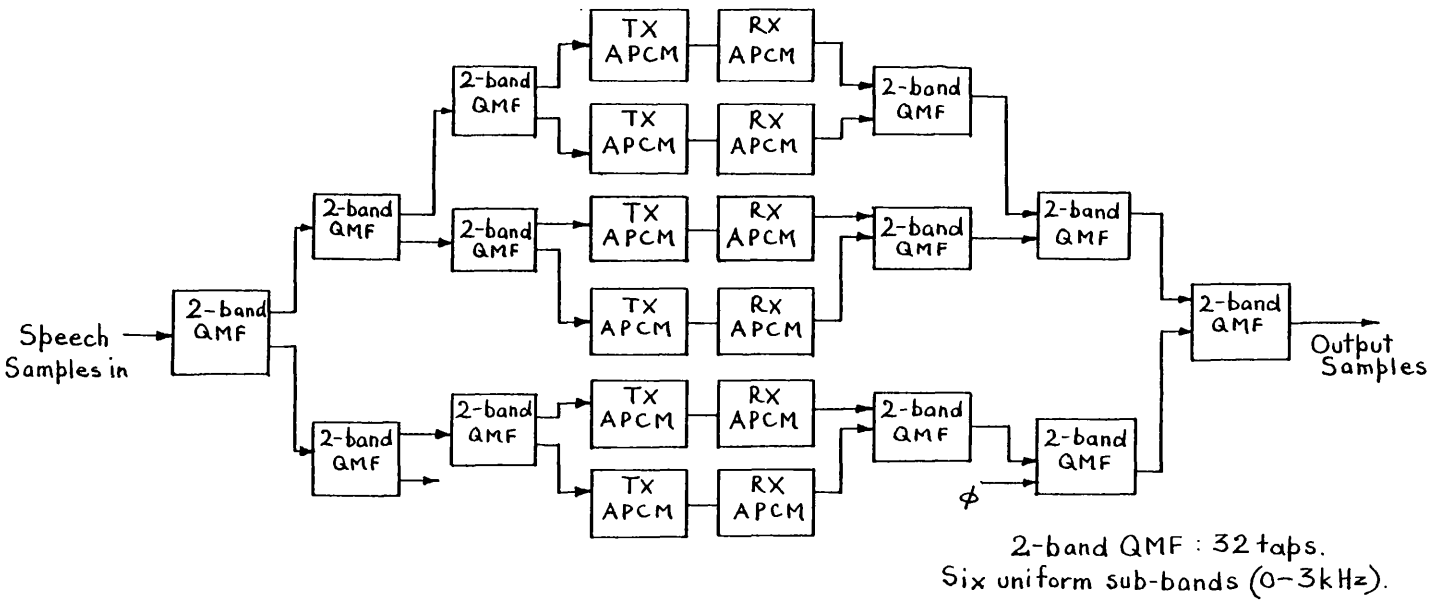


FIGURE 4.8. SUB-BAND CODER STRUCTURE IMPLEMENTED ON THE MAC68 FOR 16kb/s

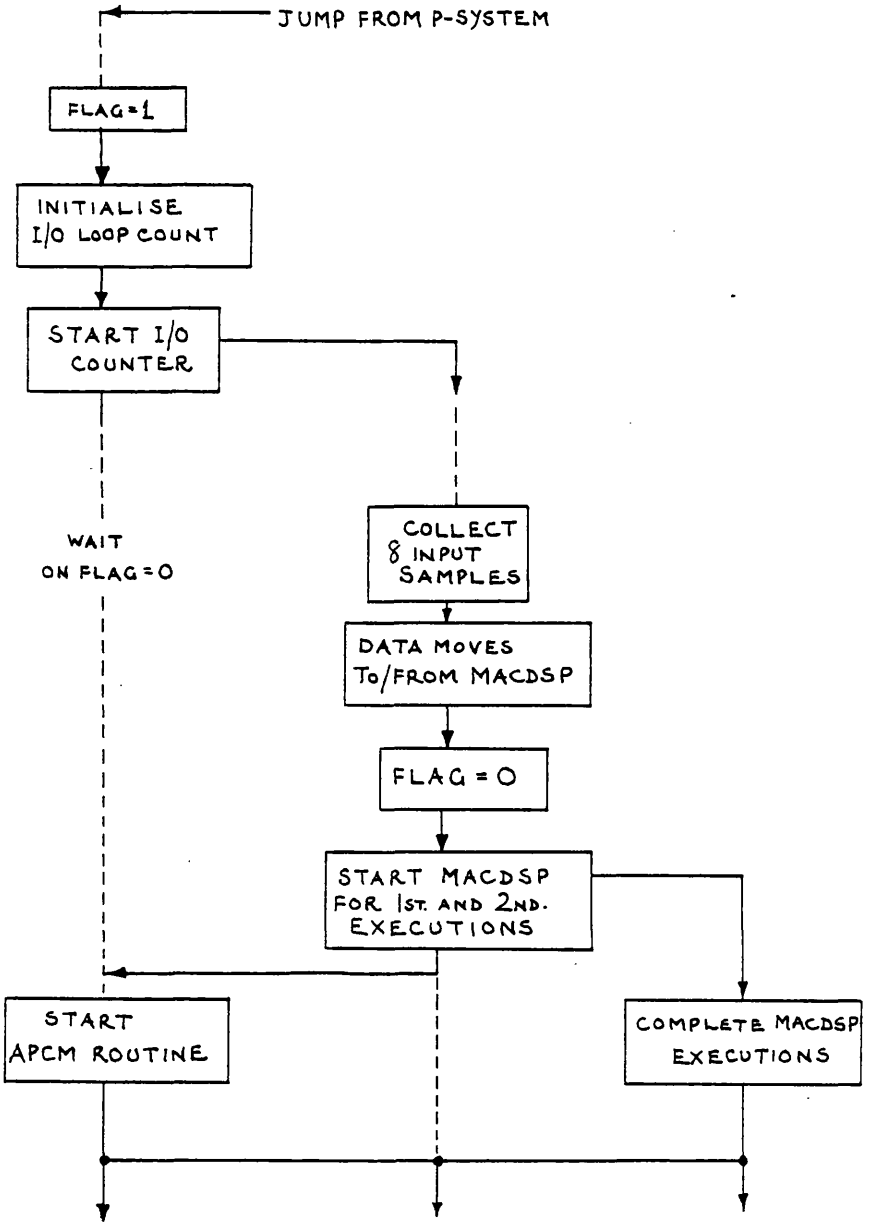


FIGURE 4.9. START-UP SEQUENCE FOR THE 16kb/s SBC.

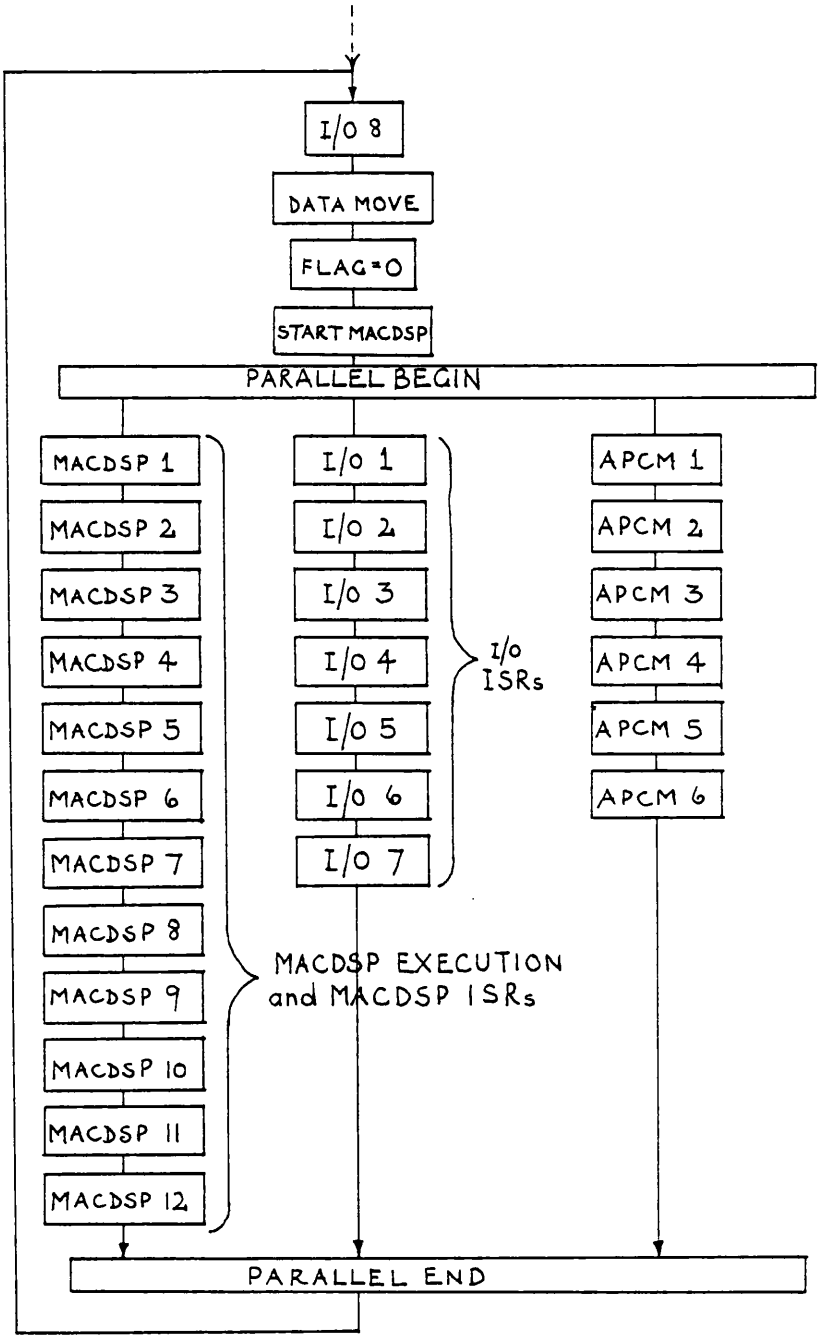


FIGURE 4.10. 'NORMAL' OPERATION OF THE 16kb/s SBC

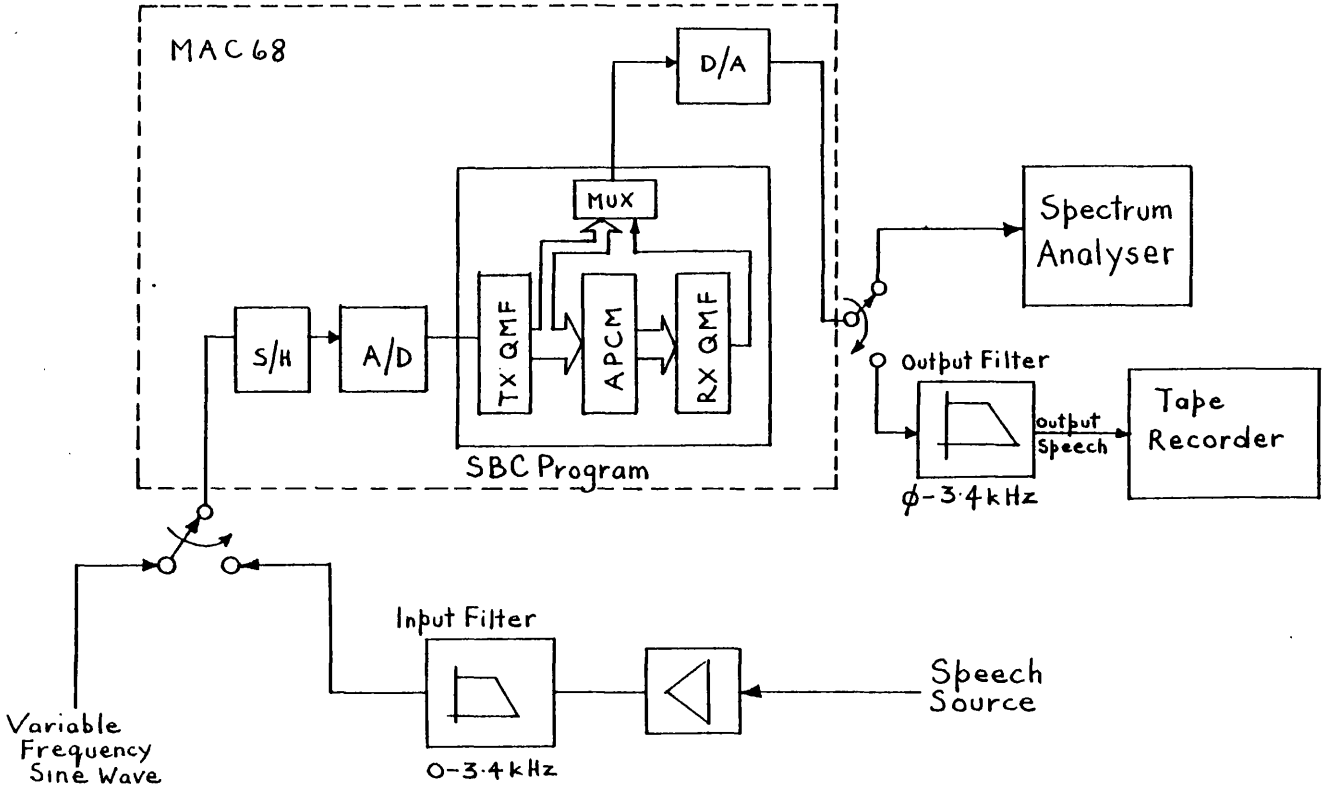


FIGURE 4.11. EXPERIMENTAL CONFIGURATION FOR THE 16 kb/s SUB-BAND CODER.

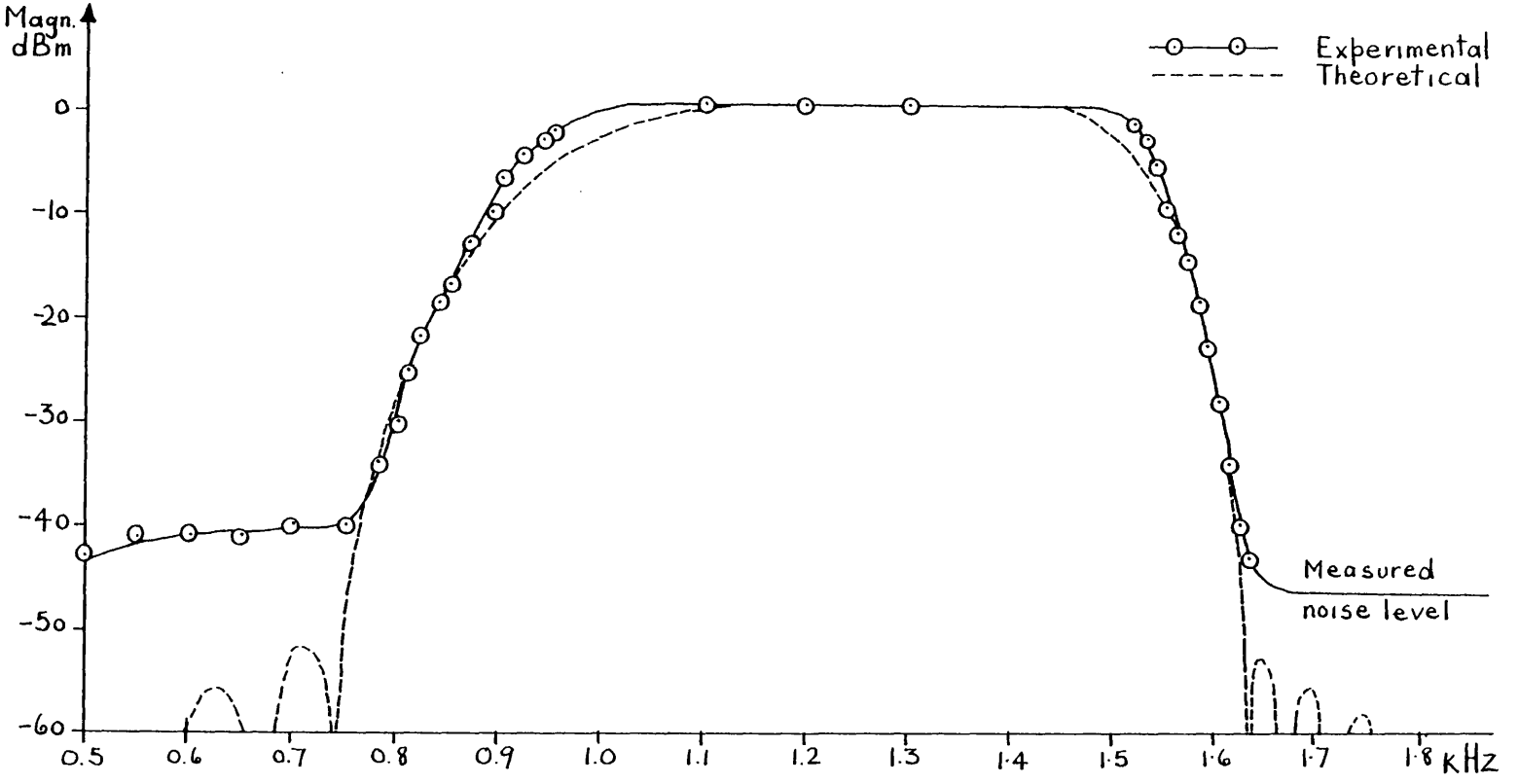


FIGURE 4.12. EXPERIMENTAL AND THEORETICAL FREQUENCY RESPONSES FOR BAND 3 (1-1.5 kHz).

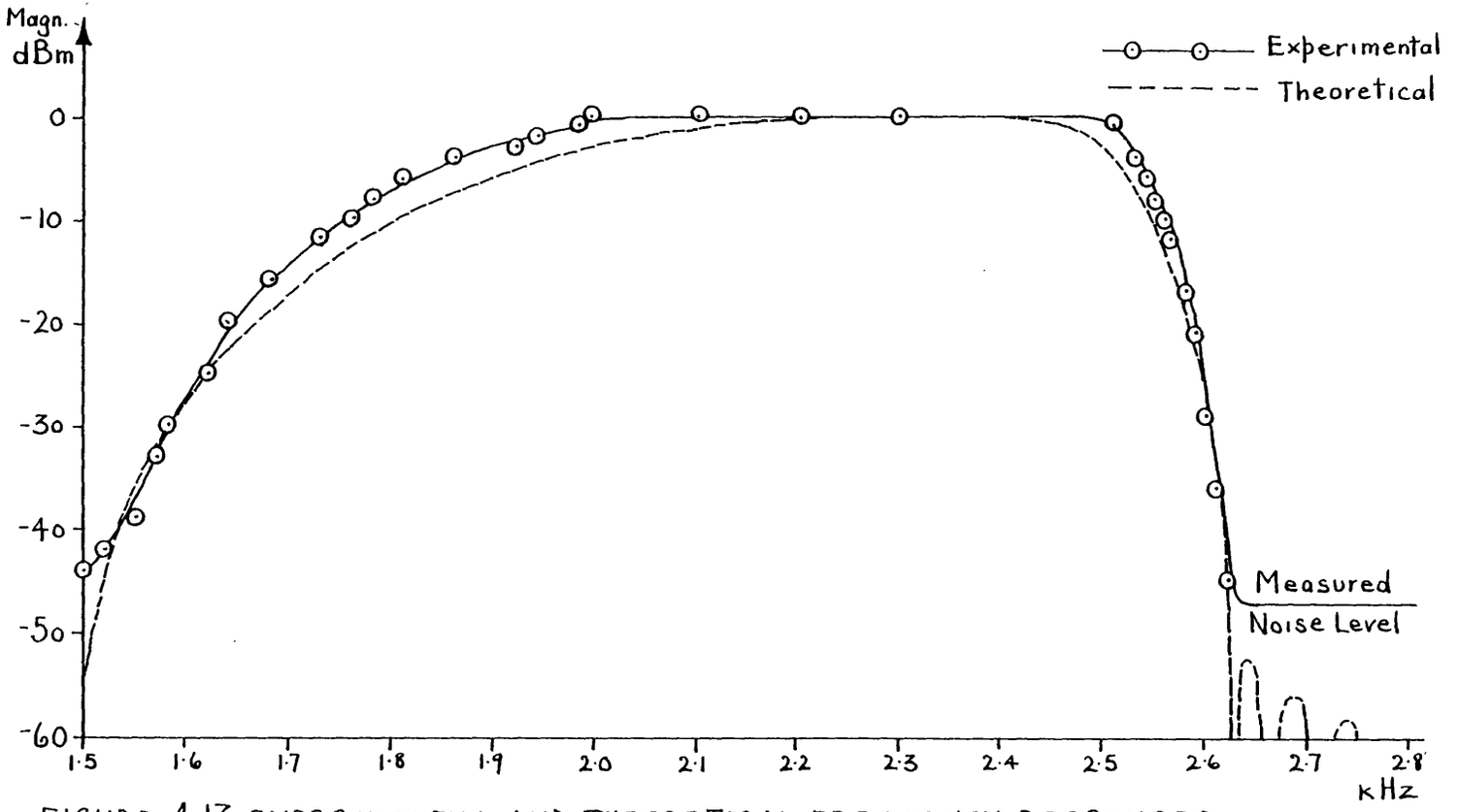


FIGURE 4.13. EXPERIMENTAL AND THEORETICAL FREQUENCY RESPONSES FOR BAND 5 (2–2·5 kHz).

B=	4	3	2
M_1	0·9	0·85	0·85
M_2	0·9	1·0	1·9
M_3	0·9	1·0	
M_4	0·9	1·5	
M_5	1·2		
M_6	1·6		
M_7	2·0		
M_8	2·4		

TABLE 4.1. APCM MULTIPLIER VALUES FOR DIFFERENT NUMBER OF BITS QUANTIZATION.

h(15) = h(16) =	·46640530E	00
h(14) = h(17) =	·12655790E	00
h(13) = h(18) =	—·99502430E	— 01
h(12) = h(19) =	—·39348780E	— 01
h(11) = h(20) =	·52947050E	— 01
h(10) = h(21) =	·14568440E	— 01
h(9) = h(22) =	—·31238620E	— 01
h(8) = h(23) =	—·41874830E	— 02
h(7) = h(24) =	·17981450E	— 01
h(6) = h(25) =	—·13036590E	— 03
h(5) = h(26) =	—·94583180E	— 02
h(4) = h(27) =	·14142466E	— 02
h(3) = h(28) =	·42341950E	— 02
h(2) = h(29) =	—·12683030E	— 02
h(1) = h(30) =	—·14027930E	— 02
h(0) = h(31) =	·69105790E	— 03

TABLE 4.2. COEFFICIENTS FOR THE 32-TAP FIR QUADRATURE MIRROR FILTER.

BAND NO.	SCALE FACTOR, A
1	1
2	0·71
3	0·35
4	0·22
5	0·16
6	0·126
	Where, $(\Delta max)_n = A_n((\Delta max)_1)$

TABLE 4.3. SCALE FACTORS USED FOR SETTING Δmax AND Δmin IN EACH SUB-BAND

5. ENHANCEMENT OF THE MAC68 SYSTEM

This chapter is devoted to the documentation of potential enhancements to the present MAC68 system. These enhancements would include the use of more advanced technologies as well as the use of a High-level language facility. This new system would therefore be faster as well as easier to program.

To justify the development of this new system, it is necessary to evaluate the present MAC68 system. Figures 5.1 and 5.2 compare the DSP performance of the MAC68 compared with other commercially available Digital Signal Computers, and Table 5.1 summarises MAC68 performance figures. As can be seen, the MAC68 approximately matches the Digital Filtering capabiity of DSP micros, and exceeds their FFT capability. These figures do not take into account Analog I/O where the MAC68 has a clear advantage, being able to execute arithmetic in parallel with complicated I/O tasks.

The MAC68 implementation of the Sub-band Coder compares favourably with the only other reported Sub-band Coder implementation using a tree-QMF structure. This other implementation uses two BELL DSP single-chip micros [C-7]. The BELL version requires the adaption of the overall Sub-band Coder task into a stream-processing form to reduce overall computation time, but use of this method may not be possible for larger block processing tasks (where I/O block management could become a bottleneck).

In general, it can be said that General-purpose tasks (for example address calculation and input-output) can be a bottleneck for DSP micros, and that 'pure' DSP tasks (for example arithmetic for digital filtering) can be a bottleneck for General-purpose micros. The MAC68 avoids these bottlenecks by combining General-purpose and DSP capabilities into a single balanced system. Consequently, the MAC68 should be suitable for a greater range of processing tasks than DSP micros or GP micros used alone. This provides the justification for continued investigations.

Having established the merit of the MAC68 architecture, ways of enhancing the present system can be sought. It is thought that the following major enhancements could be made to the MAC68:-

(1) Use of VLSI Technology.

Compaction of the current MAC68 onto a single circuit board (excluding Analog I/O) featuring a single-chip VLSI version of the MACDSP and a 12MHz MC68000.

(2) High-level Language.

Introduction of a High-level Language facility, such that code for both processors is automatically generated and linked. A mechanism must be used so that inter-processor communication is easily handled at a high-level.

Such an enhanced MAC68 system could be used in areas such as radar, sonar, telecommunications, speech processing, image processing etc.

NOTE:
This enhanced MAC68 processing unit would form a good building block for Modular Multi-processor (MM) systems (i.e. Multi-MAC68), although further discussion on this topic is beyond the scope of this current work.

The remainder of this chapter discusses the key features of the proposed enhanced system in more detail. In section 5.1 hardware enhancements are described. In section 5.2 the enhanced hardware model is used to base suggestions for High-level Language representations. Finally, in section 5.3 a full implementation example is given.

5.1 Hardware

Undoubtedly the greatest problem encountered in using the MAC68 system, concerned the 'timing' problem associated with the effective loss of a MACDSP interrupt signal when in the self-start mode. A general solution to this problem is now presented.

Consider a mechanism (in hardware or software), whereby a state variable, V, is used in the following ways:-

(1) V=0 on system start-up.

(2) A start pulse is only sent to the MACDSP when V<2, in which case V is incremented and then the start pulse sent.

(3) Each time a MACDSP sequence terminates V is decremented.

(4) MACDSP memory is only accessed by the MC68000 when V=0.

Therefore

V=0 --- MACDSP is inactive.

V=1 --- MACDSP is active, and the start buffer is not set.

V=2 --- MACDSP is active, and the start buffer is set.

Clearly, using this mechanism the self-start problem can be completely avoided. Figure 5.3 illustrates this with an example. The most efficient way to implement this mechanism is to put it into hardware, combining this hardware with other MACDSP features. This hardware would use two MC68000 addressable 1 bit flags as part of the MACDSP:-

F1 Access MACDSP memory.

F2 Send start pulse.

These flags would be read by the MC68000 prior to carrying out the named activities. A programming discipline would have to be enforced such that offset addresses, start addresses and start pulses are only sent when F2 has been set by the MACDSP control logic, and, the MACDSP memory is only accessed when F1 has been set by the MACDSP control logic.

An important point left unresolved concerns the use of the MACDSP interrupt. Using the described mechanism, the MC68000 could enter a background task while it is waiting for the MACDSP to interrupt, in which case the MACDSP ISR would switch the MC68000 back to MACDSP servicing, such that processing continues as before entering the background task. The alternative would be to have the MC68000 looping, continually testing F1 and F2 until processing can continue, this scheme would not require a MACDSP interrupt. Note that these alternatives correspond very closely to the two methods discussed in sections 4.2.1 and 4.2.2. The choice between these methods is determined primarily by ease of programming as both have their own individual merits, consequently the final decision is deferred until Section 5.2.

The current MAC68 system consists of several boards of logic, some of which is incorporated in the SAGE II micro-computer system, this is clearly undesirable for any future system. Further, some hardware inefficiencies have been allowed to creep into the MAC68 design to ease development, as evaluation of major architectural features was considered to be of greater importance than the development of an optimal hardware design. In view of the experience gained with the MAC68 system a more powerful, and more compact version can be put forward, this version would fit onto a single circuit board (using approximately 20-25 ICs), and would feature architectural enhancements (such as the one described above) and more advanced technology. The following summarises the major hardware changes to the MAC68 system.

5.1.1 MC68000 Sub-System Enhancements

The replacement of the currently used 8MHz MC68000 with a 12MHz MC68000 would be a necessary improvement. To ensure that the 12MHz MC68000 runs without using Wait states, a small amount of fast static RAM would have to be used i.e. less than 32Kbytes. Note that if less than 64Kbytes of memory is used then the absolute long addressing mode would never be used, such that only the faster absolute short addressing mode would be needed for absolute addressing. Other additional features would include a TEST ROM for system checks, I/O setup, variable initialisation and test routines (to support high-level debugging). Also, an RS232 interface to a host system would be required for downloading compiled programs, address tables, coefficients etc.

Due to the varied analog I/O requirements of different applications, the analog I/O interfaces to the MAC68 would have to be separate from the main MAC68 system features.

5.1.2 MACDSP Enhancements

The most important architectural enhancement to the MACDSP would be the incorporation of hardware to handle the above inter-processor communication mechanism. This could easily be implemented using a 2-bit up/down counter, flip-flops and random logic.

Another architectural change would involve the MACDSP instruction format and decoding. Observation of MACDSP activities shows that certain combinations of control bits will never be used, for example, loading the X register on the same cycle as storing the MAC output to data RAM. Therefore some of the MACDSP control bits can be encoded, with decoding being carried out as part of the instruction processing cycle. Four of the present control bits could be replaced by two instruction bits, C1 and C2, where:

C1	C2	
0	0	STORE MAC OUTPUT.
0	1	LOAD Y REGISTER.
1	0	LOAD X REGISTER.
1	1	LOAD X AND Y REGISTER WITH THE SAME VALUE.

Note the introduction of the new command, which would allow squaring operations to be performed (this is also possible on the current MACDSP, although it is not supported by the MACDSP Assembler). Another enhancement would be to eliminate the MAC instruction bit for clocking the MAC accumulator. Instead, the MAC accumulator instruction bits could be decoded, such that when an arithmetic operation is indicated, the MAC accumulator is clocked on the following cycle. These instruction changes concern the reduction of the instruction size. By doubling the number of offset registers to 8 there would be greater flexibility for applications such as non in-place FFTs, this though would require an additional offset register select bit.

The total number of MACDSP instruction bits would now be 20, a reduction of 4 bits over the current MACDSP. The additional delay created by the need for instruction decoding should not affect pipeline timing.

The following briefly discusses the possible features of a VLSI version of a MACDSP incorporating the above architectural changes:-

- Technology: 2 micro CMOS.
- Chip layout: Multiplier and RAM implemented as separate 'macro' blocks, all other functions implemented using rows of standard cells (using normal semi-custom formats).

- RAM: 2K*20 Program memory.
  2K*16 Data memory.
- Instruction cycle time: approximately 150nsec.
- Interface: Approximately 40 pins, interfacing directly to a 12MHz MC68000.

Note that a chip based on the MACDSP architecture could have more memory than a chip based on a DSP micro architecture (assuming the same technology for both) -- as the latter includes more functions. Having more memory on chip will increase the performance of the MACDSP relative to DSP micros for applications such as large FFTs.

5.2 High-Level Language Facility

The primary motivations for developing a High-level language facility, are as follows:

(1) The enforcement of a programming discipline for handling concurrent processes, and a general requirement for manipulating processes at an abstract level so as to increase the possibilities of finding efficient program combinations.

(2) The need to develop straight-line sequences for the MACDSP.

(3) The requirement for linking MAC68 programs to a host system (via an RS232 link), such that offset addresses and coefficients can easily be generated and linked to MAC68 variable/constant declarations.

(4) To support testing at a more abstract level, in particular the co-ordination of debugging on both the MC68000 and the MACDSP.

The main drawback in using a High-level language, is that hardware performance can be 'diluted' using compiled code as it is generally less optimal than 'hand-code'. It is thought that this inefficiency could be tolerated in view of the advantages gained.

Before discussing details of the proposed High-level language, a brief look into some fundamental issues is required. Firstly, the level of abstraction necessary to represent MAC68 programs must be decided on. Figure 5.4 illustrates fundamental levels of abstraction, with the highest level being represented by application languages, which can theoretically map down to most of the lower system languages. Many application languages are based on a functional approach which is completely independent of the underlying hardware. This is unlike systems languages, the structure of which reflect the underlying architectural characteristics, such as the sequential execution model of Von Neumann machines [B-7]. Application languages for DSP include SIPROL [G-7] and SRL [K-1], as well as some data flow languages [D-3].

The following ideas are concerned with the specification of a system language specifically for the MAC68. This particular language facility would be unsuitable for representing programs for other Digital Signal Computers. The level of abstraction possible with such a language will be just sufficient to ensure that no hand-coding is required. Also no application orientated language features will be suggested.

The first point to resolve concerns the representation of MAC68 concurrent processes. It is thought that because of the limited levels of concurrency used on the MAC68, and the complexities involved in implementing any of the currently popular concurrent programming models (see reference [A-3] for a general survey), the explicit high-level representation of concurrent processes on the MAC68 is not justifiable. Therefore, the use of background tasks (and also the use of the MACDSP interrupt) is not considered further, and all concurrency is made transparent to the programmer.

The following discussion is informal and represents an approximation to any solution which would actually be implemented.

A sub-set of the High-level language OCCAM was chosen to form the core of the proposed MAC68 language. OCCAM was chosen because it is a simple High-level language which has suitable arithmetic and control representation. Consider for example the following description of FFT bit reversal:-

```
-- 32 POINT FFT, WITH x AS THE VARIABLE TO BE BIT REVERSED AND
-- sum AS THE BIT REVERSED RESULT.
DEF N=32, M=5, Mmini=4:  -- N=M**5=NUMBER OF POINTS.
SEQ
  VAR sum:
  SEQ
    sum:=0                -- INITIALISE SUM.
    SEQ i=[0 FOR M]       -- REPEAT NEXT LINE M TIMES.
      sum:=sum+(((x>>(Mmin1-i))/\1)<<i) --'>>' AND '<<' ARE
                                        --SHIFT OPS. '/\' IS A
                                        --LOGICAL AND OPERATION.
```

A further reason for using OCCAM, is that the MAC68 concept could be extended to multiple-MAC68s, in which case the transition in software terms would be less traumatic. In these circumstances the full range of OCCAM constructs would then be used. The rest of this chapter is concerned only with the single MAC68 case.

MAC68 programs are represented by two sections, corresponding to each of the processors, where MACDSP sequences are called up from the main OCCAM text. The following details each of these separate textual sections, as well as looking very briefly at program testing.

5.2.1 MACDSP Program Representations

Figure 5.5 gives the syntactic definition for the MACDSP program representations. This definition does not include MACROs or COMMENTS, although both of these would be used in textual descriptions. This is because prior to compilation MACROs would be expanded and comments removed. The following discusses informal semantics for the possible high-level MACDSP program representations.

Declarations are used for allocating MACDSP data memory locations, the scope of these declarations covering the complete MACDSP program. Only two types are allowed, DATA and COEFF, corresponding to data variables and constants/coefficients respectively. Vectors are the only allowed form of data structure. Locations may be initialised using the INITIAL construct.

The main part of MACDSP programs are encapsulated in MODULEs. MODULE names are referenced by OCCAM calling programs, this referencing being used for accessing MACDSP locations and program sequences. For example, the IN and OUT sections allow MACDSP memory locations to be linked to OCCAM calling programs, which would reference a specific MODULE name encapsulating these sections. Note that part of a vector can be linked if required. Consider the following example for a MODULE heading:

```
MODULE name (IN data1[0..2], data2; OUT data1[0..2]);
        .
        .
END;
```

where, OCCAM values would be loaded into the pre-declared locations referred to by the IN section, and MACDSP values fetched from the pre-declared locations referred to by the OUT section. Note that MACDSP fetches by the OCCAM calling program would be carried out before MACDSP loads.

MODULES can be composed of zero or more SEQUENCES, each SEQUENCE being referenced firstly by the name of the MODULE it is contained in, and secondly by a sequence number. Each SEQUENCE on final textual expansion contains a single sequence of pseudo-instructions, all of which make explicit reference to the MAC accumulator. Consider now the semantics of these pseudo-instructions in more detail.

All identifiers used must refer to pre-declared locations, and a specialised kind of indexing, whereby offset registers are addressed explicitly, can be used. The following details some pseudo instruction examples, for which '(..)' means 'contents of':

```
( data[ $3+10 ] ); --LOAD (data.base+10+(off.reg3))
INTO THE ACCUMULATOR.

 data1[$1+2+20]*coeff[$3+1-15]+;
--ADD THE ACCUMULATOR TO THE PRODUCT OF
--(data1.base+22+(off.reg1)) AND
--(coeff.base-14+(off.reg3)).

result[10]:=*A*    --STORE THE ACCUMULATOR CONTENTS
                   --TO LOCATION (result+10).
```

A novel feature of this proposed language implementation, is the ability to start SEQUENCE executions at any labelled pseudo instruction, with execution terminating at the last MACDSP instruction in a compiled SEQUENCE. This could be carried out by using a label table for every SEQUENCE in a MACDSP program, such that an OCCAM calling program would then reference a particular start address using:

(i) MODULE name.

(ii) SEQUENCE number.

(iii) LABEL number.

Note that offset register values loaded into the MACDSP as part of an OCCAM call, would then be used in the specified SEQUENCE or SEQUENCE part. This facility is particularly useful for circular-buffering and correlation algorithms.

One further feature not mentioned, is the use of a REPEAT construct. This allows pseudo-instruction sequences to be replicatd, but with different offset address values. This construct therefore provides a macro facility as part of the language definition. Dummy variables for this construct are used in the following way:-

```
REPEAT i(1,3,2); --(START,REPITIONS,STEP)
   (x[$1+i]*y[$3+2+i])+;
END;
expands to;
(x[$1+1]*y[$3+2+1])+;
(x[$1+3]*y[$3+2+3])+;
(x[$1+5]*y[$3+2+5])+;
```

By combining MACRO and REPEAT facilities, large straight-line sequences can be generated from concise descriptions.

The mapping from pseudo-instructions to MACDSP instructions is not one-to-one, consequently compiler generated MACDSP instruction sequences must be optimised. Simple code optimisation techniques could be used for removing redundant register loads, eliminating no-operation instructions (inserted by the compiler because of the pipeline delay), as well as more complicated code optimisers for analysing variable usage [A-4]. Consider the following simple code optimisation examples:-

```
(a)   (c);     X1D,0
               Y1M,100
      (d)+;    X1D,0     --REMOVED BY OPTIMISER.
               Y1A,101
   y:=*A*      N1D,0
               P1D,102

(b)   (c);     X1D,0
               Y1M,100
   (e*f)+;     X1D,103
               Y1A,104
   y:=*A*      N1D,0     = X1D,105
               P1D,102   = P1D,102
   (g*h)-;     X1D,105   = Y1S,106
               Y1S,106   = NEXT INSTRUCTION.
```

where c,d,y,e,f,g,h are locations 100,101,102,103,104,105,106 respectively, and location 0 contains +1.0.

Code optimisers used must ensure that labelled pseudo instructions are correctly mapped into labelled MACDSP instructions, and that execution started at any labelled instruction conforms to the original semantics (this is a pre-requisite of any code optimiser). Also optimisation should be applied after program testing (see Section 5.2.3), so that pseudo instruction operations can be isolated.

5.2.2 OCCAM Calling Program Representations

Some parts of the OCCAM definition that would not be included for calling program representations are, input/output processes, channels, parallel/alternative processes and character constants. The sub-set would then be adapted to accommodate the various system interfaces required for the MAC68. The following details the major additions to the sub-set:

(a) MACDSP Interface.

Specialised statements would be required for accessing MACDSP memory and controlling MACDSP execution. These statements could be indicated with '*', so that MACDSP access statements could have the form:-

```
*ACCESS module.name (IN data1, OUT data2)
```

This statement could link to the corresponding MODULE input/output statements in the MACDSP program representation. Statements for controlling MACDSP execution would be more complicated, for example:

```
*module.name [seq.numb,label.numb]
($1:=expression,$2:=...)
```

where a MACDSP sequence would be addressed as described in section 5.2.1 and offset register values to be used in the addressed MACDSP sequences would be calculated using normal compiled OCCAM expressions prior to their being loaded into the relevant offset registers ('$' being used to indicate offset register numbers for both OCCAM and MACDSP representations). These special-purpose statements would ensure that MC68000 to MACDSP communication conventions are adhered to. When calling MACDSP sequences, the following MC68000 assembler sequence could be used:

```
LOOP:  TST    F2
       BEQ    LOOP          ; WAIT FOR F2=1.
       MOVE   VALUE1,OFF.REG1  ;  LOAD  OFFSET REG. 1.
       MOVE   VALUE2,OFF.REG2  ;  LOAD  OFFSET REG. 2.
         .
         .
       MOVE   VALUE3,ST.ADDR   ;   LOAD   START ADDRESS.
       MOVE   --,START     ; SEND START PULSE.
```

A similar convention would be used for ACCESS statements except execution would wait on the F1 flag.

(b) I/O Statements.

Specialised statements would also be needed for handling I/O interfaces. For example the scope of an I/O statement could determine which sections of an OCCAM program are to be executed concurrently with I/O processing, and real-time execution could then continue as for an OCCAM WHILE TRUE statement. The I/O statement would therefore be placed such that real-time execution is initiated once all coefficients and addresses had been loaded, and variables initialised i.e. part of the compiled I/O statement would start up the I/O hardware. Consider for example the statements:

```
...INITIALISATION STATEMENTS...
io(IN data1[10], OUT data2[10])
SEQ
   ...MAIN PROGRAM TO BE EXECUTED CONCURRENT
      WITH I/O...
```

where the compiled I/O Interrupt Service Routine would be responsible for synchronising the main program to the I/O streams, formatting I/O values and maintaining data buffers (such as the data1 and data2 vectors in this example).

To minimise the MC68000 switching times between the I/O ISR and the main program, MC68000 registers could be allocated for the use of the I/O ISR only.

(c) Program Headings.

By using a physical data link to some form of host facility, coefficient values and complicated address tables could be calculated using a data processing language such as PASCAL or FORTRAN. These values could then be downloaded into the MAC68 real-time environment. The host could link these values to particular MAC68 memory locations via a special form of program heading, for example:

```
program name(coeff.table[50], addr.table[100])
```

where the 150 address and coefficient values would be loaded prior to real-time execution. This facility would enable the MAC68 system to be completely integrated into a much larger programming environment, therefore speeding up program development dramatically.

Aspects of compiler implementation are beyond the scope of this discussion, which provides a framework only.

5.2.3 Program Testing

Both of the processors used in the MAC68 system, the MC68000 and the MACDSP, can be single-stepped, therefore a hardware foundation is available for building any form of high-level testing facility. The following briefly discusses a potential candidate for such a facility.

The test facility could take on the form of a high-level monitor, whereby labels could be used in both OCCAM and MACDSP program representations to indicate display points (similar to breakpoints in microprocessor debuggers). Program testing could then be carried out prior to final preparations of code for real-time execution i.e. MACDSP execution would be sequential with MC68000 execution and code optimisation would be carried out once test labels had been removed.

The name of variables to be displayed would be inputted to the test package, as well as the names of labels at which a display is required. The actual display would be composed of two halves corresponding to the two processors. Note that the MAC accumulator value could be displayed also, as for the present MACDSP debugger.

5.3 Example

To illustrate the potential of an enhanced MAC68, the following details the programming of a 16 sub-band tree-QMF structure (for the Transmit side only, although it is thought that the Receive structure and waveform coders and decoders could also be implemented). None of the timing problems encountered in Chapters 3 & 4 are present for this enhanced system, further, as MACDSP interrupts are not used, computational structures no longer need to be grouped together to reduce overall inter-processor communication overheads.

The actual tree-QMF structure consists of four stages, where stages are:-

Stage 1: 64 tap 2-band QMF.

Stage 2: 32 tap 2-band QMFs.

Stages 3 & 4: 16 tap 2-band QMFs.

Figure 5.6 details the high-level language program. The following briefly summarises the main features:-

(a) Circular-Buffering

The proposed circular-buffer scheme would require 2 MACDSP cycles (at 200nsec per cycle) for every filter tap, this being without the increased data memory requirements detailed in Chapters 3 & 4.

This is because each of the 2-band tree-QMF building blocks; can be broken down into separate SEQUENCES, and each component part can then be executed separately one or more times to give the required overall structure. Note that the flexibility of this method relies on MAC register contents remaining unchanged between MACDSP start-ups. For this example, sum-of-products SEQUENCEs would be executed twice to give the required circular-buffer effect, with partial product-sums being held in the MAC accumulator between MACDSP executions.

(b) Offset Addresses.

Sixteen input data samples would firstly be loaded into the MACDSP using a compiled ACCESS statement, the tree-QMF structure would then be executed in the order of the data flow through the tree. Note that two variables would be needed for calculating offset addresses for pointing to QMF delay variables. Further, the OCCAM calling program would use a table of pre-calculated offset addresses for pointing to 2-band QMF output locations.

(c) Approximate Execution Time.

Assuming that 2msec (corresponding to 16 samples at 8kHz sample frequency) is available for executing a complete 16-band structure, then the MACDSP execution time for Transmit and Receive tree-QMFs (assuming optimal code generation) would then be approximately 50% of this time.

In general, this DSP structure is quite complex compared to most other DSP tasks, so that in being able to illustrate this structure other structures should be more easily representable.

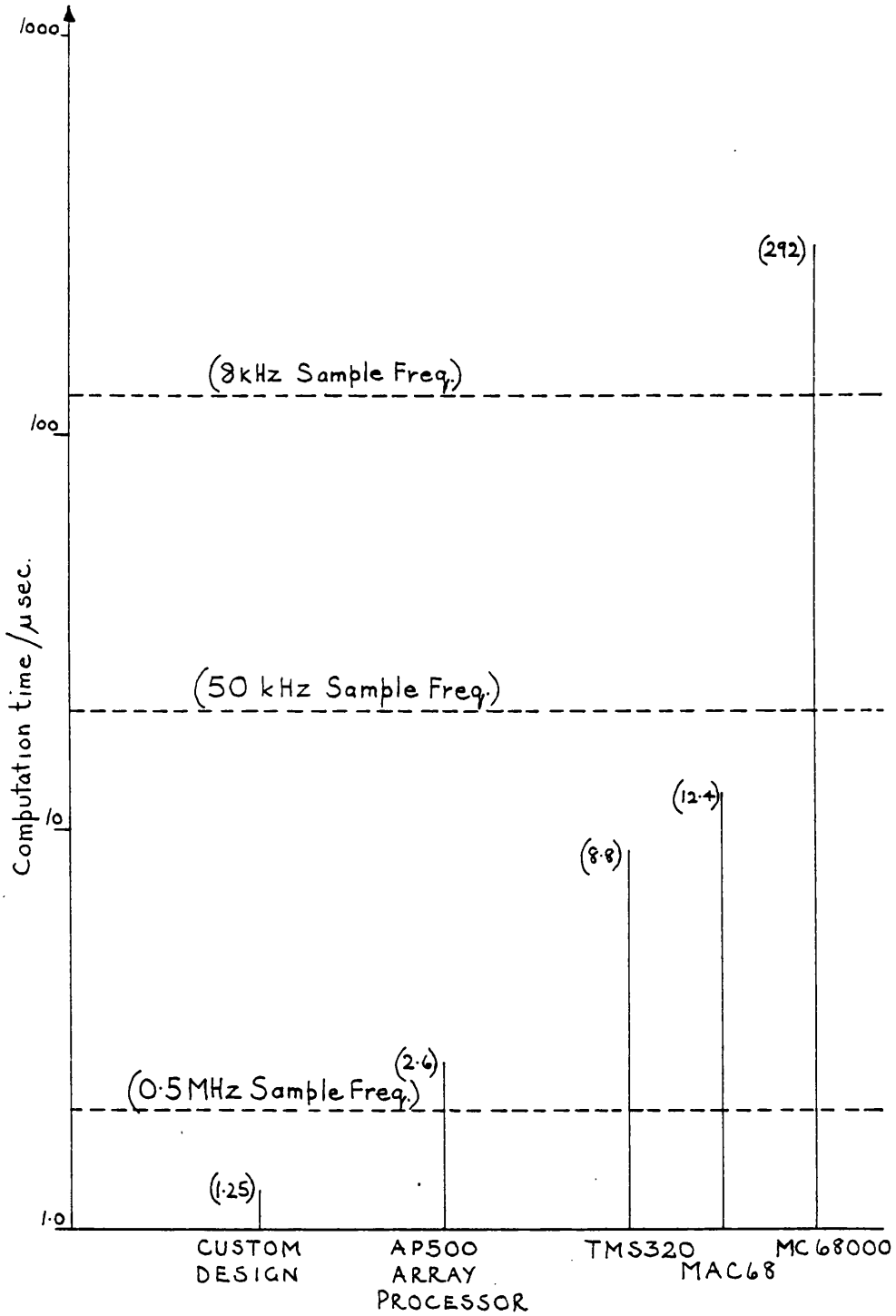


FIGURE 5.I. APPROXIMATE COMPARISON OF MAC68 PERFORMANCE FOR AN 8TH. ORDER DIGITAL FILTER (as for Appendix I, Benchmark I).

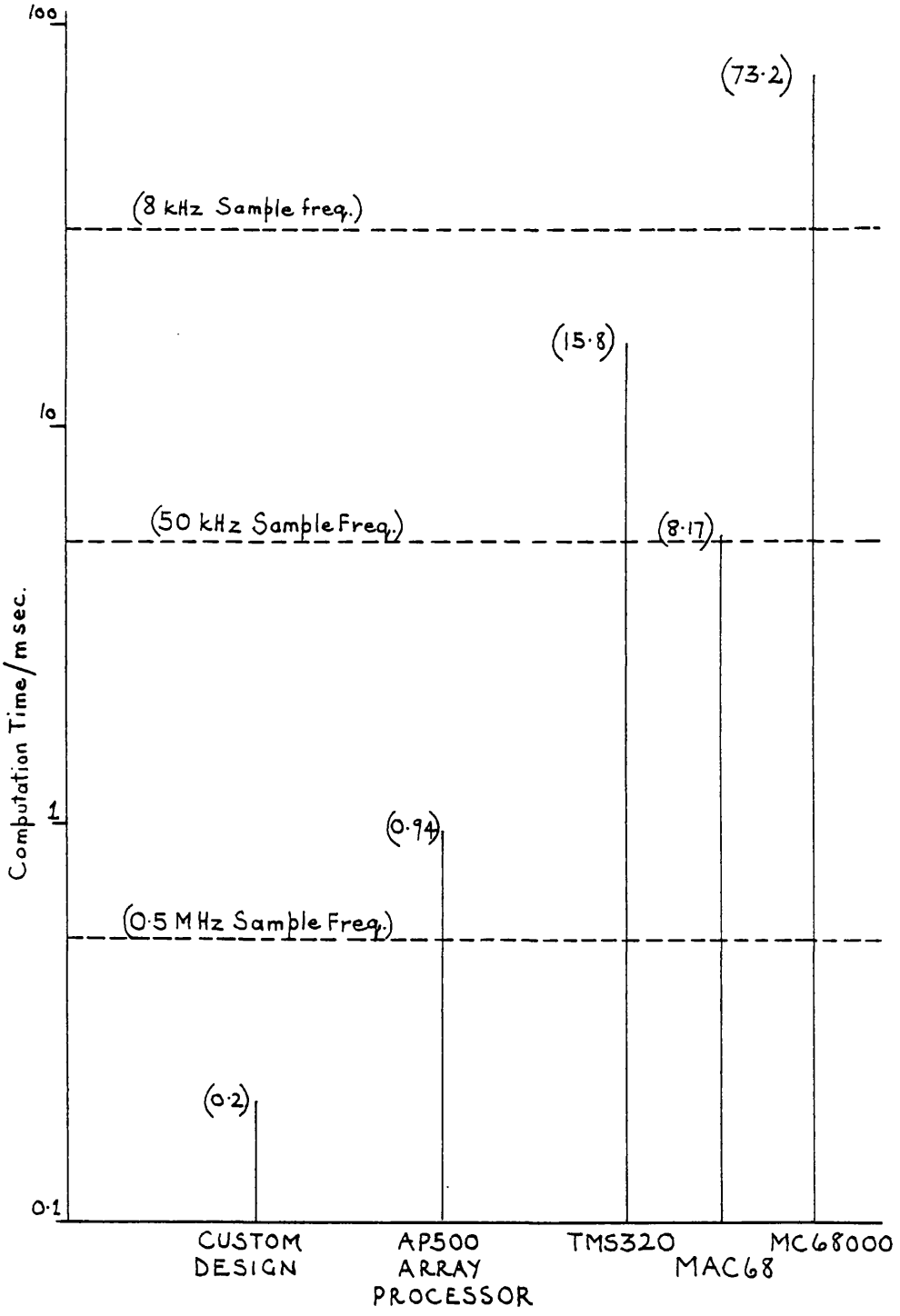


FIGURE 5.2. APPROXIMATE COMPARISON OF MAC68 PERFORMANCE FOR A 256PT. COMPLEX FFT.

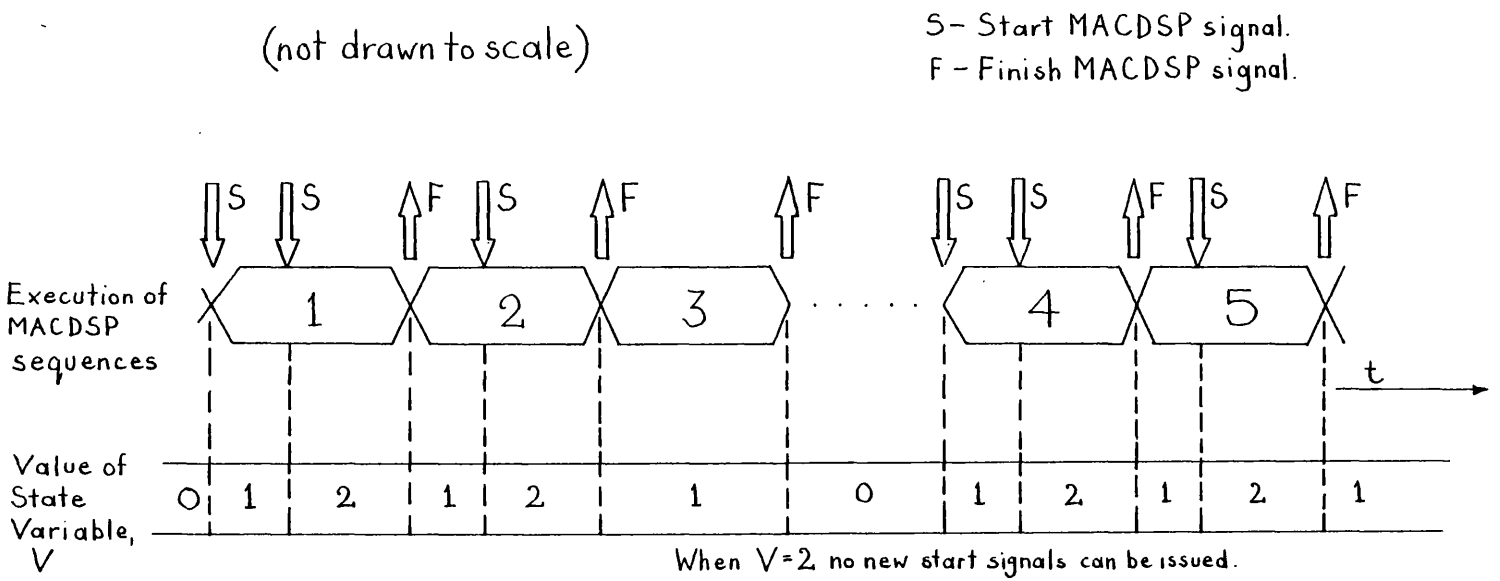


FIGURE 5.3. USE OF STATE VARIABLE, V, FOR CONTROLLING COMMUNICATION BETWEEN MC68000 AND MACDSP.

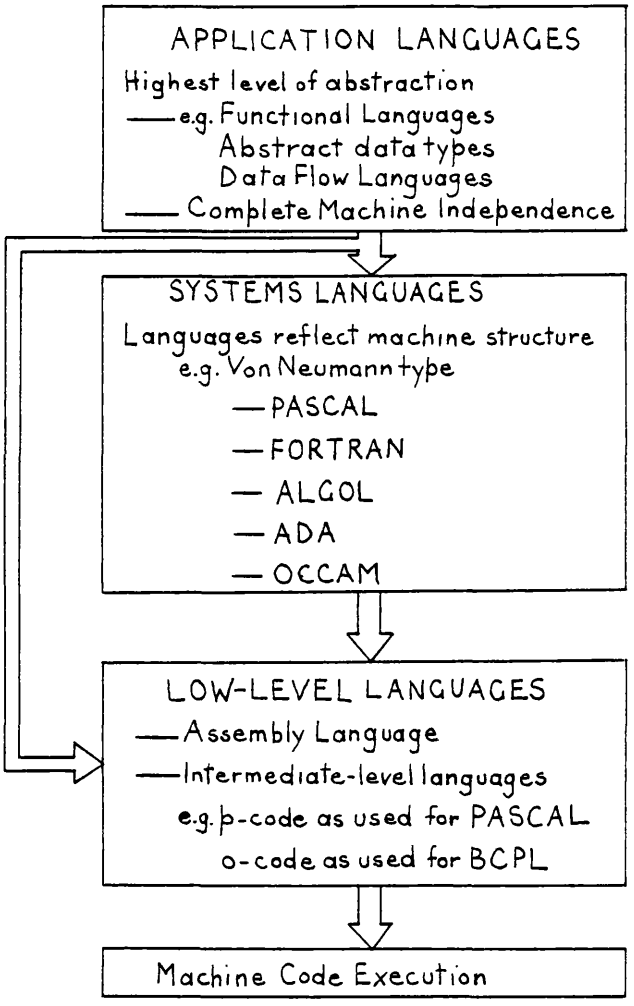


FIGURE 5.4. POSSIBLE LEVELS OF ABSTRACTION FOR PROGRAMMING LANGUAGES.

SYMBOLS:

< > SYNTACTIC CATEGORY.

{ } REPEATED ZERO OR MORE TIMES.

| OR.

[ ] OPTIONAL.

(NOTE: ' IS USED TO DISTINGUISH [AND]
IN THE PROGRAM TEXT FROM THE SAME
CHARACTERS USED TO REPRESENT
OPTIONAL TEXT ie. '[' WOULD APPEAR
IN THE ACTUAL PROGRAM TEXT)

<program>= MACDSP <globals> <main> MACDSP-END

<globals>= <data.dec> [<coeff.dec>] [<init>]

<data.dec>= DATA <dec.list> ;

<coeff.dec>= COEFF <dec.list> ;

<init>= INITIAL <assign> { , <assign> } ;

<dec.list>= <reference> { , <reference> }

<assign>= <reference> := <initial>

<initial>= [<sign>] <digit> | <real.constant>

<real.constant>= £ [<sign>] [<digit>] . <number>

<reference>= <identifier> ['[' <number> ']']

<main>= <module> { ; <module> }

<module>= MODULE <identifier> [(<in.out>)] ;
[<block>] END

<in.out>= [<ins>] [; <outs>]

<ins>= IN <link> { , <link> }

<outs>= OUT <link> { , <link> }

Figure 5.5 Syntax for proposed MACDSP language
(continued on next page).

```
<link>= <identifier> [ '[' <range> ']' ]

<range>= <number> | <number> .. <number>

<block>= <sequence> { ; <sequence> }

<sequence>= SEQUENCE '[' <value.sum> ']' <commands> END

<commands>= <command> { ; <command> }

<command>= <repeat> | <instructions>

<repeat>= REPEAT <dummy.dec> { , <dummy.dec> } ;
                                    <instructions> END

<dummy.dec>= <identifier> ( <parameters> )

<parameters>= <parameter> , <parameter> [ , <parameter> ]

<parameter>= [ <sign> <value> ]

<instructions>= <inst.line> { ; <inst.line> }

<inst.line>= [ <value.sum> : ] <instruct>

<value.sum>= <value> { <sign> <value> }

<instruct>= <assignment> | <accum.action>

<assignment>= <loc> := *A*

<accum.action>= ( <loc> [ * <loc> ] ) [ sign ]

<loc>= <identifier> [ '[' <first.value> { <sign> <value> } ']' ]

<first.value>= $ <value> | [ sign ] <value>

<value>= <identifier> | <number>

<sign>= + | -

<number>= <digit> { , <digit> }

<identifier>= <letter> { <letter> | <digit> }

<digit>= 0 | 1 | .. | 9

<letter>= a | b | c| .. | y | z
```

Figure 5.5 continued.

```
MACDSP
DATA x[320],temp_up,temp_down,temps[80];
COEFF h64[64],h32[32],h16[16],zero;
INITIAL zero:=0;
        --SET VALUES FOR ALL COEFFICIENTS

        MACRO qmf(p1,p2,p3);
          SEQUENCE [p1]
            (temps[$2]);
            x[$1]:=*A*;
            (zero)
          END;
          SEQUENCE [p1+1]
            REPEAT i(0,p2), j(1,p2,2);
              i: (x[$1+i]*p3[$2+j])+;
            END;
            p2+1: temp_up:=*A*
          END;
          SEQUENCE [p1+2]
            (temps[$1+1]);
            x[$1+p2]:=*A*;
            (zero)
          END;
          SEQUENCE [p1+3]
            REPEAT i(0,p2), j(0,p2,2);
              i: (x[$1+i+p2]*p3[$2+j])+;
            END;
            p2+1:temp_down:=*A*
          END;
          SEQUENCE [p1+4]
            (temp_up);
            (temp_down)-;
            temps[$2]:=*A*;
            (temp_up);
            (temp_down)+;
            temps[$1]:=*A*
          END
        END;

MODULE qmfs (IN temps[0..15]; OUT temps[64..79] );
  qmf(0,32,h64);
  qmf(5,16,h32);
  qmf(10,8,h16);
END;

MACDSP--END
```

Figure 5.6 High-level Language program for a 16 Sub-band tree-QMF structure (continued on next page).

```
DEF N1=32, N2=16, N3=8:
DEF lev1.base=0, lev2.base=64, lev3.base=128:
DEF lev4.base=196:
DEF offset= TABLE[ BYTE 17,25,18,26,19,27,20,28,21,29,22,
30,23,31,24,32,33,37,34,38,35,39,36,40,41,45,42,46,43,47,
44,48,49,51,50,52,53,55,54,56,57,59,58,60,61,63,62,64,65,
66,67,68,71,72,69,70,77,78,79,80,75,76,73,74]:

      PROC level(VALUE N,s,n,dat.off,in.out)=
      VAR par1,par2,par3,par4:
      SEQ
        par1:=n+dat.off
        *qmf[s](#1:=par1,#2:=in.out)
        par2:=-(n<<1)+dat.off
        *qmf[s+1,n](#1:=dat.off,#2:=par2)
        par3:=N-n
        par4:=(n-N)+dat.off
        *qmf[s+1,par3](#1:=par4,#2:=dat.off)
        *qmf[s+2](#1:=par1,#2:=in.out+1)
        *qmf[s+3,n](#1:=dat.off,#2:=par2)
        *qmf[s+3,par3](#1:=par4,#2:=dat.off)
        *qmf[s+4](#1:=offset[BYTE in.out],
                  #2:=offset[BYTE in.out+1])

VAR n1,n2,n3,n4:
SEQ
  n1:=0
  n2:=0
  n3:=0
  n4:=0
  io(input,output)
  SEQ
    *ACCESS qmfs(IN data.in)
    VAR in.out,dat.off:
    SEQ
      in.out:=0
      dat.off:=lev1.base
      SEQ i=[0 FOR 8]
        SEQ
          n1:=n1-1
          IF n1<0
            n1:=N1-1
          level(N1,0,n1,dat.off,in.out)
          in.out:=in.out+2
```

Figure 5.6 continued.

```
  SEQ j=[0 FOR 4]
    SEQ
      n2:=n2-1
      IF n2<0
        n2:=N2-1
        dat.off:=lev2.base
        SEQ k=[0 FOR 2]
          SEQ
            level(N2,5,n2,dat.off,in.out)
            in.out:=in.out+2
            dat.off:=dat.off+(N2<<1)
  SEQ j=[0 FOR 2]
    SEQ
      n3:=n3-1
      IF n3<0
        n3:=N3-1
        dat.off:=lev3.base
        SEQ k=[0 FOR 4]
          SEQ
            level(N3,10,n3,dat.off,in.out)
            in.out:=in.out+2
            dat.off:=dat.off+(N3<<1)
  SEQ
    n4:=n4-1
    IF n4<0
      n4:=N3-1
      dat.off:=lev4.base
      SEQ k=[0 FOR 8]
        SEQ
          level(N3,10,n4,dat.off,in.out)
          in.out:=in.out+2
          dat.off:=dat.off+(N3<<1)
*ACCESS qmfs(OUT data.out)
```

Figure 5.6 continued.

Single FIR tap (with a circular buffer)		0·5 μsec.
Bi-quad filter section (with a circular buffer).		3·0 μsec.
N word cross-correlation	N = 32 N = 64 N = 128 N = 256	0·56 msec. 2·114 msec. 8·384 msec. 33·152 msec.
N point Complex FFT.	N = 32 N = 64 N = 128 N = 256	0·37 msec 0·924 msec 2·216 msec 5·168 msec

TABLE 5.1. MAC68 PERFORMANCE FIGURES

6. CONCLUSIONS

A novel classification of Real-time Programmable Digital Signal Processors or Digital Signal Computers (DSC) was developed and is illustrated in Figure 1.3. The S-type (single instruction stream) classification was used in the study of the Digital Signal Processing (DSP) performance of General-purpose (GP) and DSP micros (Table 2.1).

Figures 2.1 - 2.3 illustrate the range of DSP performance found with GP micros, in which the MC68000 appears best overall. It was found that the use of a hardware multiplier device, the TRW MPY-16, does not greatly improve the DSP performance of the later 16-bit GP micros, the speed-up being approximately 1.4 in the case of the MC68000. The study showed that DSP micros are significantly faster than GP micros in executing common DSP tasks. For example, comparing an MC68000 + MPY-16 with the TMS32010:

Digital Filters:- TMS32010 approx. 20 times faster

Small-size FFTs:- TMS32010 approx. 5 times faster
($\leq$ 64 points)

This speed advantage appears to be obtained via the use of special-purpose hardware such as an array multiplier, with architectural features such as pipelining and separate data and program memories. However, general-purpose tasks such as index address calculation or input-output block management can form processing bottlenecks with DSP micros.

A novel DSC system, the MAC68, which balances general-purpose and arithmetic capabilities, was developed. The MAC68 is a Functional Multi-processor (FM(2)) and uses two independent processors, the fastest benchmarked GP micro, the MC68000, and a custom designed pipelined arithmetic processor, the MACDSP. The MACDSP is based around the TDC1010J Multiplier-Accumulator, and has an instruction cycle time of 250nsec. The matching of these two processors gives the combined system all-round abilities. Figures 3.1 and 3.2 give high-level block diagrams of system hardware (which includes an analog I/O sub-system). To aid application development, software support packages were developed, including a MACDSP assembler and debugger, and a MAC68 loader/linker. Figures V.1 and VII.1 illustrate the interaction and use of these software components. Syntax for the MACDSP Assembly language is given in Figure 3.3.

A 16Kb/s Sub-band Coder was successfully implemented on the MAC68. This Sub-band Coder features six sub-bands with Adaptive Pulse Code Modulation in each band, and its complexity compares favourably with the only other similar Sub-band Coder reported in [C-7]. Figure 4.8 gives the overall Sub-band Coder structure implemented, and Figures 4.7, 4.9 - 4.10 illustrate how the concurrent tasks execute. The final task configuration was arrived at after careful study of the basic alternatives, in particular those for circular and non-circular buffering (see Figures 4.4 - 4.7). Figures 4.12 - 4.13 give the measured frequency responses for two of the six sub-bands.

The Sub-band Coder application fully verified the MAC68 concept, in that it combined intensive arithmetic tasks such as digital filtering, with general-purpose tasks such as I/O block management, as well as a full utilisation of the system parallelism.

Figures 5.1 and 5.2 give approximate comparisons of MAC68 DSP performance with that of other Digital Signal Computers. The MAC68 approximately matches the Digital Filtering capability of DSP micros, and exceeds their FFT capability:

8th order digital filter:-	(	TMS32010	8.8usec
	(	MAC68	12.4usec
256 point complex FFT:-	(	TMS32010	15.8msec
	(	MAC68	8.17msec

The MAC68 therefore greatly exceeds the performance of a single MC68000, the corresponding MC68000 figures being 292usec and 73.2msec respectively. In general, because the MAC68 has all-round capability, it performs better than DSP micros for tasks which mix general-purpose and arithmetic operations. In particular note that the MAC68 can perform I/O tasks completely in parallel with the execution of data arithmetic (a fact which was not taken into account for Figures 5.1 and 5.2).

Proposals are put forward for the immediate enhancement of the MAC68 system. These include the incorporation of a new mechanism to aid the utilisation of parallel processing, also suggestions for MACDSP implementation as a single-chip, and the use of a High-level Language. Figure 5.6 gives the proposed High-level Language program for a 16 sub-band tree QMF structure. It is thought that this enhanced MAC68 system would be very competitive on the Digital Signal Computer market (especially when the system is based on an industry standard microprocessor, the MC68000).

Without question the future in DSC architectures will be towards greater and greater parallelism. A possible direction for such advances could be to have:-

(a) Algorithm concurrency exploited at the highest level via the use of Modular multi-processor (MM) structures.

(b) Algorithm concurrency exploited at the lowest level via the functional partitioning of individual MM building blocks.

MAC68-type architectures are likely candidates for the building blocks of (b), as these building blocks must be capable of being used as "stand-alone" processors, and the MAC68-type architecture has been shown (in this Thesis) to have advantages over S-type architectures. MM systems utilising MAC68-type building blocks would therefore be Modular Functional-Multi-processors.

Future work should now concentrate on the design of Functionally partitioned architectures which simultaneously match the requirements of DSP algorithms and VLSI technology. This work would include studies of the relationships between different types of functions in DSP algorithms, such that parallelism can be successfully identified, and appropriate functional units designed. This work must also include studies on the impact of architectural features on programming, otherwise additional performance will be acquired at the expense of programmability.

APPENDIX I

BENCHMARK ALGORITHMS.

Three computational structures representing a typical cross-section of Digital Signal Processing were chosen. Benchmarks for each structure were then derived.

Benchmarks generally reside between two extremes, one extreme using a minimum of memory but involving a maximum of address calculation and control, and the other extreme using a minimum of address calculation and control but a maximum of memory. An important point that should be made here, is that, minimum computation does not necessarily imply minimum execution time for a particular architecture. This is because for some architectures, Von Neumann architectures in particular, it might be faster to calculate an address operand than to fetch a pre-calculated address operand from memory.

Implementation of the derived benchmarks varies for different architectures. Therefore benchmark algorithm descriptions should be treated as approximations.

Sixteen bit operands are used for benchmarks I and II, and 8 bit operands for benchmark III. This introduces an element of variation in the dynamic range of the arithmetic used in the benchmarks, such that the effect on computation times can be studied.

I.1 BENCHMARK I: Eight Order Digital Filter

The computational structure for this benchmark is four cascaded scond-order sections (bi-quad sections), and is derived from the algorithms presented in [N-1]. The basic equations for each section are given by:

$$M(k) = x(k)+t1 \qquad (I.1)$$

$$y(k) = a0.m(k)+t2 \qquad (I.2)$$

$$t1= -b1.m(k-1)-b2.m(k-2) \qquad (I.3)$$

$$t2= a1.m(k-1)+a2.m(k-2) \qquad (I.4)$$

where

x(k) is the section input value.

y(k) is the section output value.

m(k), m(k-1) and m(k-2) are delay-line variables.

t1 and t2 are temporary variables.

a0, a1, a2, b1 and b2 are section coefficients.

Each section therefore requires 5 multiplications, and the complete filter 20 multiplications.

NOTE:

Second-order sections are often implemented with only 4 multiplications, in the above case this would mean a0=1.

Figure I.1 gives the benchmark algorithm description. This algorithm is structured to minimise input-output delay, also coefficients are assumed pre-scaled such that overflow cannot occur and no data scaling is assumed. A drawback of designing coefficients so that overflow cannot occur, is that the full dynamic range of the number representation may be under-utilised for some classes of input signal [0-1].

I.2 BENCHMARK II: 256-Point Complex Fast Fourier Transform (FFT).

The FFT structure chosen for this benchmark, has the following characteristics:-

(a) Complex data and coefficients.

(b) 256 data points.

(c) Radix-2 butterfly.

(d) In-place computation.

(e) Decimation-in-time.

The total number of butterfly computations is 1024, each butterfly being represented by the following equations:-

$$Xm+1(q) = Xm(p)-Xm(q).C(r) \quad (I.5)$$

$$Xm+1(p) = Xm(p)+Xm(q).C(r) \quad (I.6)$$

where

Xm(p), Xm(q) are complex data inputs to the butterfly.

Xm+1(p), Xm+1(q) are complex data outputs from the butterfly.

C(r) is the complex coefficient used in the butterfly.

In practice these equations are implemented with the complex product being carried out first. Therefore each butterfly requires 4 multiplications, giving a total of 4096 multiplications for the complete FFT structure.

Figure I.2 gives the benchmark algorithm description. Two pre-calculated address offsets are used for each butterfly executed. Also the real and imaginary values for each butterfly output are multiplied by 0.5 to prevent overflow [W-1].

I.3 BENCHMARK III. 32-Word Cross-Correlation

The computation structure for this benchmark is based primarily on sum-of-products sequences, which are represented by the following equations (where N=32):-

$$
\begin{aligned}
r(-N+1) &= y(0).x(N-1) \\
r(-N+2) &= y(0).x(N-2)+y(1).x(N-1) \\
&\vdots \\
r(0) &= y(0).x(0)+y(1).x(1)+\ldots\ldots+y(N-1).x(N-1) \\
&\vdots \\
r(N-2) &= y(N-2).x(0)+y(N-1).x(1) \\
r(N-1) &= y(N-1).x(0) \qquad \text{(I.7)}
\end{aligned}
$$

where

Y= y(0),y(1)...y(N-1) - input vector.

X= x(0),x(1)...x(N-1) - input vector.

R= r(-N+1),..,r(0),..r(N- 1) is the output vector.

The total number of multiplications for this benchmark is 1024. Figure I.3 gives the benchmark algorithm description, which uses two nested control loops.

```
var m0,m1,m2,t1,t2:array[1..4];
    a0,a1,a2,b1,b2:array[1..4];
    y;
    i,j,k;

 begin
    initialise;
    input(y);
    for i:=1 to 4 do
    begin
      m0[i]:=y+t1[i];
      y:=a0[i]*m0[i]+t2[i];
    end;
    output(y);
    for j:=1 to 4 do
    begin
      m2[j]:=m1[j];
      m1[j]:=m0[j];
    end;
    for k:=1 to 4 do
    begin
      t1[k]:=-b1[k]*m1[k]-b2[k]*m2[k];
      t2[k]:=a1[k]*m1[k]+a2[k]*m2[k];
    end;
  end;
```

Figure I.1. BENCHMARK ALGORITHM I:
8TH ORDER DIGITAL FILTER.

```
var X:array[0..511];
    C:array[0..255];
    addr_table:array[0..2047];
    i,j,bfly_count,data1_addr,data2_addr;
    coeff_addr,sep;

              procedure bfly( p,q,r);
              var Rz,Iz;
              begin

                { EACH DATA/COEFF VALUE IS }
                { TWO LOCATIONS. REAL }
                { FOLLOWED BY IMAGINARY VALUE.}

                Rz:=X[q]*C[r]-X[q+1]*C[r+1];
                Iz:=X[q+1]*C[r]+X[q]*C[r+1];
                X[q]:=(X[p]-Rz)/2;
                X[q+1]:=(X[p+1]-Iz)/2;
                X[p]:=(X[p]+Rz)/2;
                X[p+1]:=(X[p+1]+Iz)/2;
              end;
  begin
    initialise;
    sep:=2;
    bfly_count:=0;
    for i:=1 to 8 do    { 8 PASSES }
    begin
      for j:=1 to 128 do
      begin
        bfly_count:=bfly_count+2;
        data1_addr:= offset_table[bfly_count];
        coeff_addr:= offset_table[bfly_count+1];
        data2_addr:= data1_addr+sep;
        bfly(data1_addr,data2_addr,coeff_addr);
      end;
      sep:=sep*2;
    {SEPERATION OF BFLY.INPUTS DOUBLES EACH PASS}
    end;
  end;
```

Figure 1.2. BENCHMARK ALGORITHM II:
256-POINT COMPLEX FAST FOURIER TRANSFORM.

```
const initial_res,finish,start;
var   addr1,addr2,addr3,count;
      sum;
      x,y:array[0..31];
      r:array[0..63];

{THE FOLLOWING IS ONLY ONE HALF OF }
{THE BENCHMARK.THE OTHER HALF IS }
{ALMOST IDENTICAL.}

   begin
     res_addr:=initial_res;
     addr1:=finish;
     for count:=1 to 32 do
     begin
       sum:=0;
       addr2:=start_addr+count;
       for j:=0 to count do
       begin
         sum:=sum+x[addr1]*y[addr2];
         addr1:=addr1-1;
         addr2:=addr2-1;
       end;
       r[res_addr]:=sum;
       res_addr:=res_addr+1;
     end;
   end;
```

Figure I.3. BENCHMARK ALGORITHM III:
32-WORD CROSS-CORRELATION.

APPENDIX II

MC68000 LOGIC DESCRIPTION

The MC68000 system components used are embedded in a SAGEII microcomputer system (this saved on hardware and software development). Figure II.1 shows a condensed block diagram of these components. Important features are as follows:-

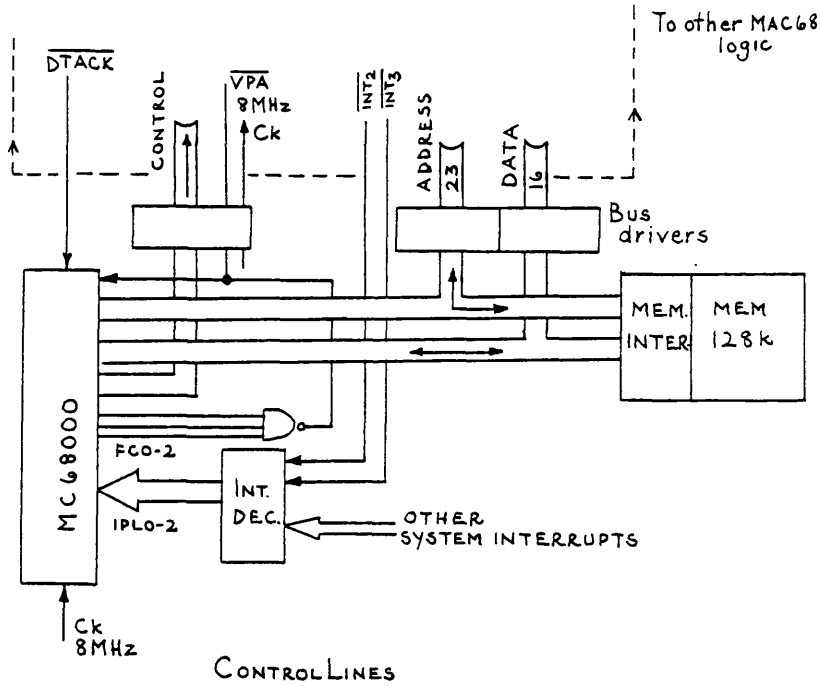
1. MC68000 hardware interrupts--Priorities of 2 and 3 are hardwired for the MACDSP and analog I/O respectively and autovectoring is used instead of data vectoring. When the MC68000 is interrupted, the VPA- line is used to reset the interrupt latch on either the MACDSP or the analog I/O module, this is done by putting the interrupt priority on address lines A1-3 so that the correct interrupt latch is reset.

2. Control lines--RW-, LDS-, AS- and DTACK- are used to control memory. Logic is provided on both the MACDSP and on the analog I/O module to ensure that DTACK- is lowered when either module is addressed, otherwise the MC68000 would jump to an internal TRAP vector. The MC68000 8MHz clock is sent to both modules, this saves on clock logic.

3. Memory--Memory is 128K of dynamic RAM. Because of the fast access time of this RAM, the MC68000 is able to run at its maximum speed with zero WAIT states (although there is a small time penalty for refresh cycles).

NOTE:

Signal names followed by '-' denote active low signal lines. This convention is also used in the following Appendices.



Control Lines
- RW
- LDS and UDS
- AS

Other Signals
- 16 Bit Data Bus
- 23 Bit Address Bus
- 8 MHz Ck
- $\overline{VPA}$
- $\overline{DTACK}$
- $\overline{INT_2}$
- $\overline{INT_3}$

FIGURE II.1. INTERFACE LOGIC BETWEEN THE MC68000 AND OTHER MAC68 UNITS.

APPENDIX III

MACDSP LOGIC DESCRIPTION

Figure 3.2 shows the complete MACDSP system diagram. The diagram is split into 9 functional sub-blocks, each of which is described in the following sections. Capital letters are used to signify functional components.

Detailed MACDSP timing examples are given in Figure III.10.

III.1 Decoder Logic (Figure III.1).

The upper MC68000 address lines are used to select the MACDSP, and lower address lines (A13-A15) are used to select units on the MACDSP. Output lines from the decoder logic are as follows:-

ST.LTCH.ENB Used to latch the MACDSP start address from the MC68000 address bus.

COUNT.LOAD- Enables the PROGRAM COUNTER load logic.

START.ENB- Sets the MACDSP START BIT.

OFFREG.ENB- Used to generate a write pulse to the OFFSET REGISTER banks.

INST1.ENB- Used for writing the 1st half of a MACDSP instruction to PROGRAM RAM.

INST2.ENB- Used for writing the 2nd half of a MACDSP instruction to PROGRAM RAM.

DATA.ENB- Used for reading/writing to DATA RAM.

Output(2) of the decoder is used for activating the ST.LTCH.ENB and COUNT.LOAD- lines at the beginning of a program load sequence (only possible when the MACDSP is not active). Output(3) of the decoder is used for de-activating these lines, this corresponding to the termination of a program load sequence. The ST.LTCH.ENB line is also activated by output(1) when a MACDSP start address is loaded into the START ADDRESS LATCH.

Because all locations addressed by the MC68000 must return DTACK- (to avoid a MC68000 BUS ERROR), the MACDSP.SEL line is gated with AS- onto the DTACK- line. Therefore, every-time the MACDSP is selected DTACK- is activated.

III.2 Start Address Latch and Program Counter (Figure III.2)

Prior to a PROGRAM RAM load, ST.LTCH.ENB and LOAD- must be activated to make both the START ADDRESS LATCH and PROGRAM COUNTER transparent. This effectively connects the MC68000 Address bus to the PROGRAM RAM address pins, so that a program load can proceed.

During normal operation, ST.LTCH.ENB can be used to latch the MACDSP start address from the MC68000 address bus. This may occur at any time independent of the MACDSP being active, so that a new start address can be latched to give effective double-buffering if required i.e. the old start address having been already loaded into the PROGRAM COUNTER.

The LOAD- signal is used during normal operation for loading the start address from the START ADDRESS LATCH into the PROGRAM COUNTER. The start address loaded into the PROGRAM COUNTER can be the same as for the previous program execution, or can have been changed for a different program.

Counting is initiated by the ACT signal.

III.3 Offset Registers and Address Adder (Figure III.3)

Two banks of 4 registers (12 bits each) are used. At any one time, one register bank is used for MACDSP operation, and the other register bank is available for accepting values from the MC68000 data bus (two banks are used to give effective double buffering). The BANK SW signal is used for switching the register banks over, this occurs at the beginning of a MACDSP program execution sequence, so that values loaded into registers prior to a MACDSP start are used during the subsequent program sequence.

Registers are written into by firstly selecting a register using MC68000 address lines (A1-2), and then generating a write pulse with LDS-, OFFREG.ENB- and a BANK SWITCH output.

Two control bits REGSEL(2), are outputted from the PROGRAM RAM on every MACDSP instruction cycle, and used for selecting a register for reading (the output of the other bank being held at a high impedance). The register output value (OFFSET ADDRESS) is then added to the MAIN ADDRESS, outputted from the PROGRAM RAM on the same cycle as the REGSEL(2) bits, and then the address sum (DATA ADDRESS) is latched into TEMP LATCH on the next rising MACDSP CK edge. DATA ADDRESS is then available for addressing the DATA RAM during the next MACDSP cycle.

To ensure that the output of TEMP LATCH cannot corrupt MC68000 addresses when the MACDSP is inactive and the MC68000 is accessing the DATA RAM, the ACT signal is connected to the TEMP LATCH output control.

III.4 Program Memory and Buffers (Figure III.4)

Up to 1024 MACDSP instructions, each 24 bits long, can be stored in PROGRAM RAM. Because the width of the MC68000 data bus is 16 bits, MACDSP instructions must be loaded in two halves of 12 bits each.

Gating is designed so that the data buffer outputs are only enabled when the MC68000 is selecting the PROGRAM RAM and the MACDSP is inactive.

The PROGRAM RAM select signals INST1.CS- and INST2.CS- are continually active when the MACDSP is active. These signals are also used during a program load sequence.

Not all of the program memory is of the same type, as the REGSEL(2) bits must be of outputted earlier than the other bits. Therefore these two bits use 70ns RAM and the other bits use 100ns RAM. This faster RAM is needed to account for the propagation delay between the OFFSET REGISTERS and the ADDRESS ADDER.

III.5 MAC and Data RAM Timing Logic (Figure III.5)

There are a total of 8 control bits, all of which directly control MACDSP resources. The control bits can be split into 3 fields:-

END bit (CONTROL 8)

DATA R/W- bit (CONTROL 7)

MAC CONTROL bits (CONTROL 1-6)

The END bit is used to terminate a MACDSP sequence (see CONTROL section). The DATA R/W- bit is used for generating the read/write pulse for the DATA RAM (this logic is designed so that when the MACDSP is active the MC68000 R/W- signal has no affect).

Because CONTROL1 and CONTROL7 are both used during a MACDSP write cycle, they are clocked out on a rising MACDSP.CK edge. For example, TSM- controls the MAC output to the DATA RAM, so that the entire write operation must occur between MACDSP.CK edges. CONTROL 2-6 are clocked out on a falling MACDSP CK edge, this is to ensure that they are valid for the next rising MACDSP.CK edge, when the MAC instruction and operands are loaded into the MAC.

III.6 Data RAM and Associated Logic (Figure III.6)

The MACDSP data memory (DATA RAM) consists of 1024 words of 16 bits each, with transfers to the MC68000 being one word at a time. Because the MC68000 buses are effectively disconnected from the DATA RAM when the MACDSP is active, accessing of DATA RAM can only be done when the MACDSP is inactive.

In the first quarter of a MACDSP write cycle, the two most significant bits of the MAC LSP are enabled onto the DATA BUS, and are latched at the end of this quarter cycle, therefore during this period the DATA RAM output is disabled. The DATA RAM is then only selected for the last three-quarters of a MACDSP cycle.

III.7 Multiplier-Accumulator(MAC) and Associated Logic

(Figure III.7)

The MAC logic is designed so that the accumulator value is available one MACDSP.CK cycle after the operands have been clocked in.

The MAC is configured for

- Two's complement arithmetic (TC=1)
- Truncation of the output value (RND=0)
- No preloading (PREL=0)
- XTP buffer enabled (TSX=0)

The rising MACDSP.CK edge is used for clocking the following:-

- Operands into the X register or the Y register
- MAC instructions

  NOTE:

  The MAC instruction register is clocked with the OR of the X register and Y register clocks.
- The MAC accumulator

Because the Y input and the LSP use the same pins internally, and the Y input pins are connected to the lower 14 MSP pins external to the MAC, the MSP and LSP outputs cannot be enabled at the same time. Therefore these outputs are enabled separately, with the LSP output being enabled for the 1st quarter cycle (of a MACDSP write cycle) so that the 2 relevant LSP bits are latched. For the remainder of the cycle this 2-bit latch output and the 14 MSP bits

all enabled onto the DATA BUS (see Figure III.10). Therefore the required 16bit value is written to DATA RAM.

The MAC.CK.ENB- signal ensures that the MAC can only be clocked when designated by a MACDSP instruction, so that any change in the MAC internal state conforms to the semantics of the MACDSP program. Clocking of the MAC when not designated by the MACDSP program would corrupt MAC register/ accumulator values, and so give incorrect results when single stepping.

When MAC.CK.ENB- is active, CONTROL2-4 are used to enable clock edges to the MAC. Clock edges are applied to the following:-

CLKX-Loads a 16bit value from the DATA BUS into the X register.
CLKY-Loads a 16bit value from the DATA BUS into the Y register.
(NOTE: CLKX and CLKY are also used for loading the MAC instruction).
CLKP-Loads the arithmetic result into the accumulator.

CONTROL5 and CONTROL6 are connected to the SUB and ACC inputs of the MAC, which are used for determining MAC arithmetic operations.

III.8 Control Logic (Figure III.8)

An RC circuit and push button switch are used to generate the RESET signal, normally carried out immediately after power up. This RESET signal ensures that:-

(1) The START BIT is reset.

(2) ACT=0, such that MACDSP RAM can be accessed by the MC68000.

(3) The INTERRUPT BIT is reset.

Assuming that the MACDSP has been reset and loaded with all relevant programs and parameters, it can then be started. This is done by sending a pulse on START.ENB-. The start-up sequence is then as follows:-

(1) START.ENB- sets START BIT=1.

(2) The OFFSET REGISTER banks are switched, so that pre-loaded values can now be used.

(3) The LOAD signal is activated, so that the value held in the START ADDRESS LATCH is loaded into the PROGRAM COUNTER.

(4) ACT is set to logic 1, therefore starting the PROGRAM COUNTER and effectively disconnecting all MACDSP RAM from the MC68000 buses.

(5) The START BIT is reset to logic 0.

(6) MAC.CK.ENB- is activated to allow clocking of the MAC where designated by MACDSP instructions.

The MACDSP has been designed with a SELF-START mode. In this mode the MACDSP can be immediately re-started (going through sequence 2-6 above) when END becomes active with the last MACDSP instruction in a sequence. To enable SELF-START, the START BIT must be set before END becomes active, so that parameters and START.ENB- are sent while the MACDSP is active.

The MAC.CK.ENB- logic is designed so that the MAC can only be clocked two MACDSP.CK cycles after the ACT signal has changed to logic 1, and never when ACT=0, therefore ensuring that MAC values cannot be corrupted between MACDSP program sequences. When a MACDSP instruction with END=0 is executed, the program sequence is terminated. Normally only the last instruction in a sequence has END=0, and for single-stepping all instructions have END=0. The END signal generated from the END bit, is delayed by the correct amount to ensure that the above process works correctly.

III.9 Interrupt Logic (Figure III.9)

The INTERRUPT BIT is set by the ACT- signal when a MACDSP program sequence terminates. When this happens the INTERRUPT(2) signal is activated and, if the MC68000 is executing a program of priority lower than 2, it will then start an autovector sequence. As part of this sequence, the priority of the interrupt will be put onto address lines A1-3, and the function code lines FC0-2 will indicate INTERRUPT ACKNOWLEDGE. Using a comparator and decoding logic, these lines are used to reset the INTERRUPT BIT. After the interrupt acknowledge cycle the MC68000 will execute the

priority(2) interrupt routine, the start address of which is held in the autovector table.

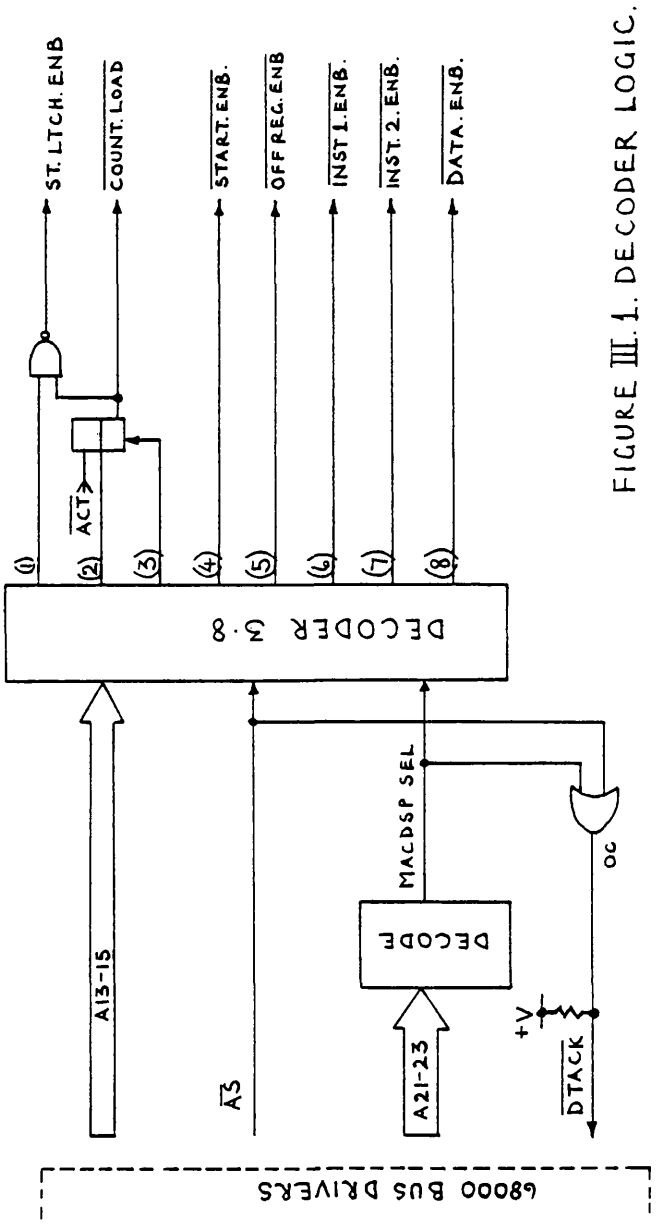


FIGURE III.1. DECODER LOGIC.

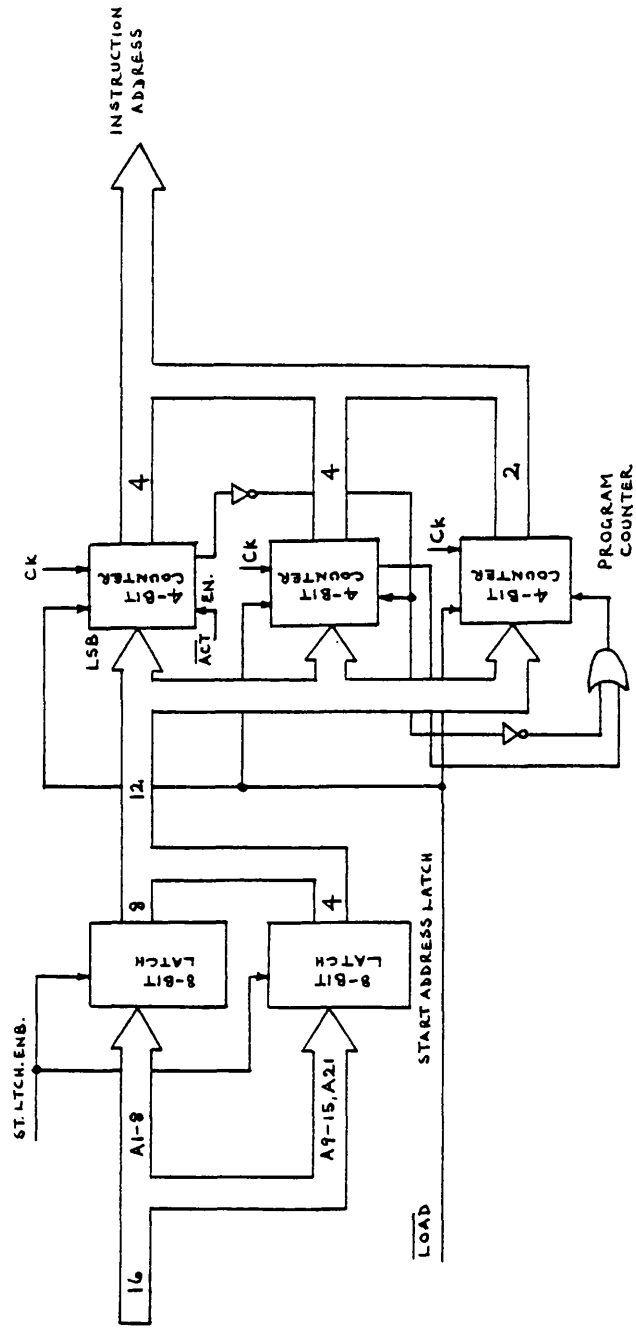


FIGURE III. 2. START ADDRESS LATCH AND PROGRAM COUNTER.

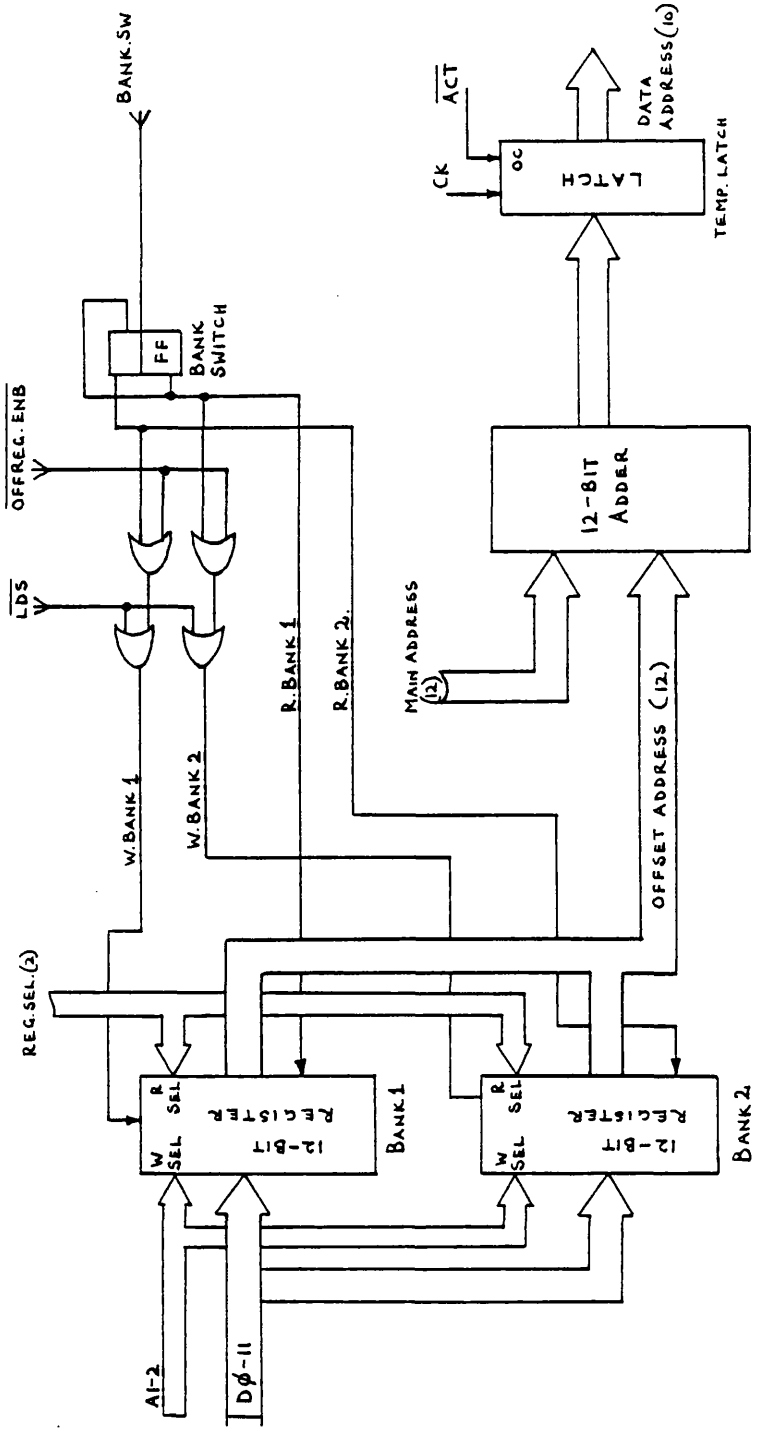


FIGURE III 3. OFFSET REGISTERS AND ADDRESS ADDER.

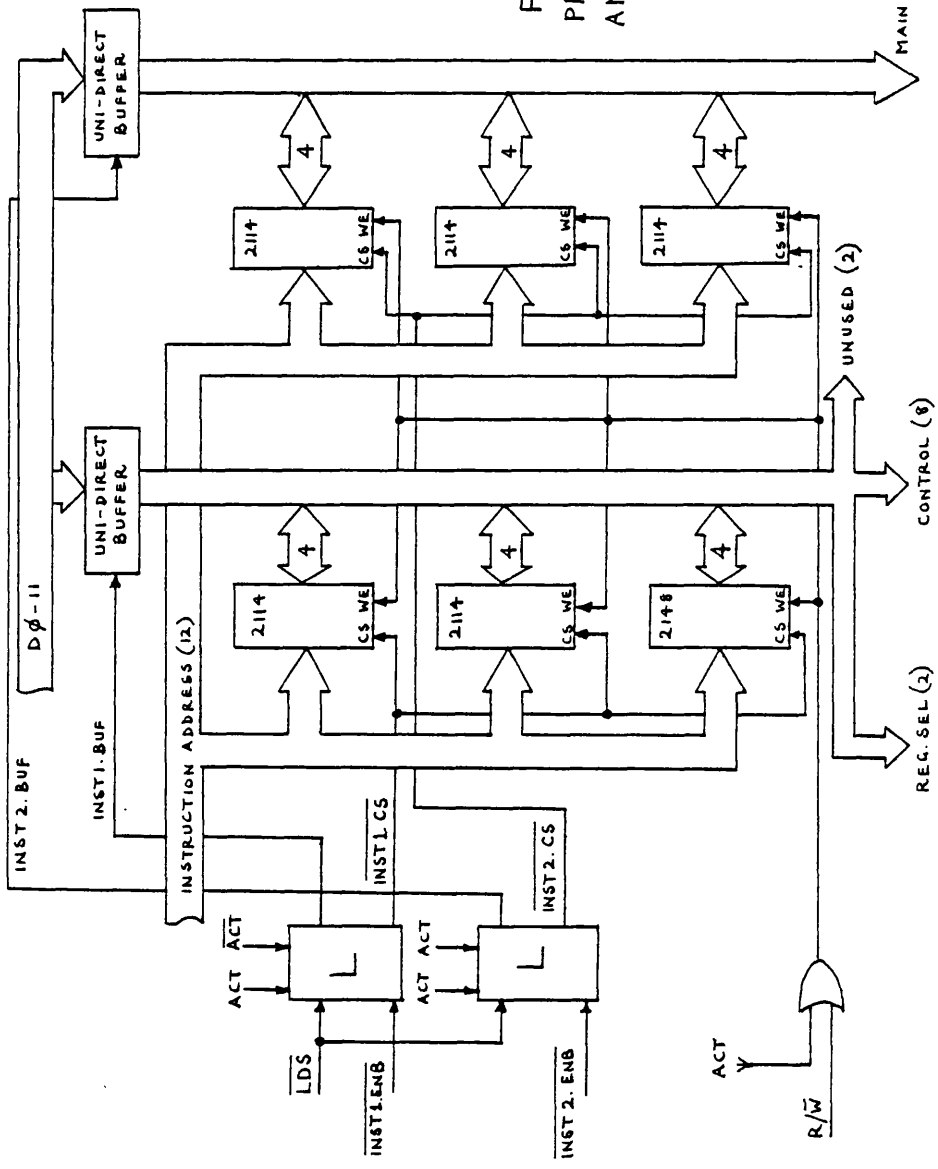


FIGURE III.4.
PROGRAM MEMORY
AND BUFFERS.

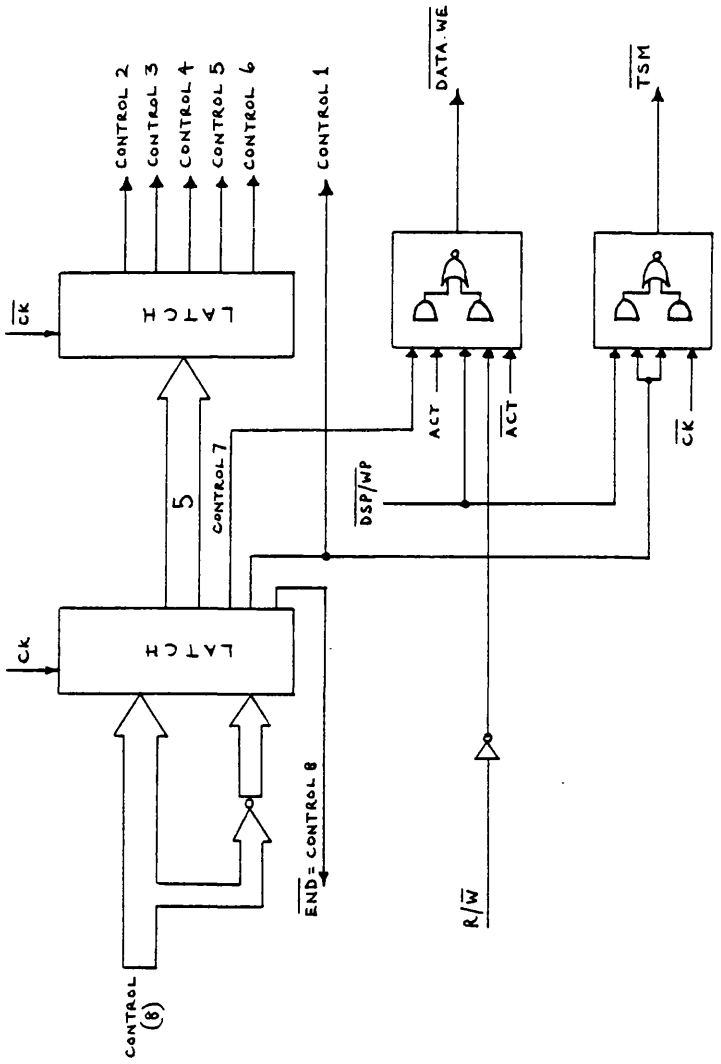


FIGURE III.5. MAC/DATA RAM TIMING LOGIC.

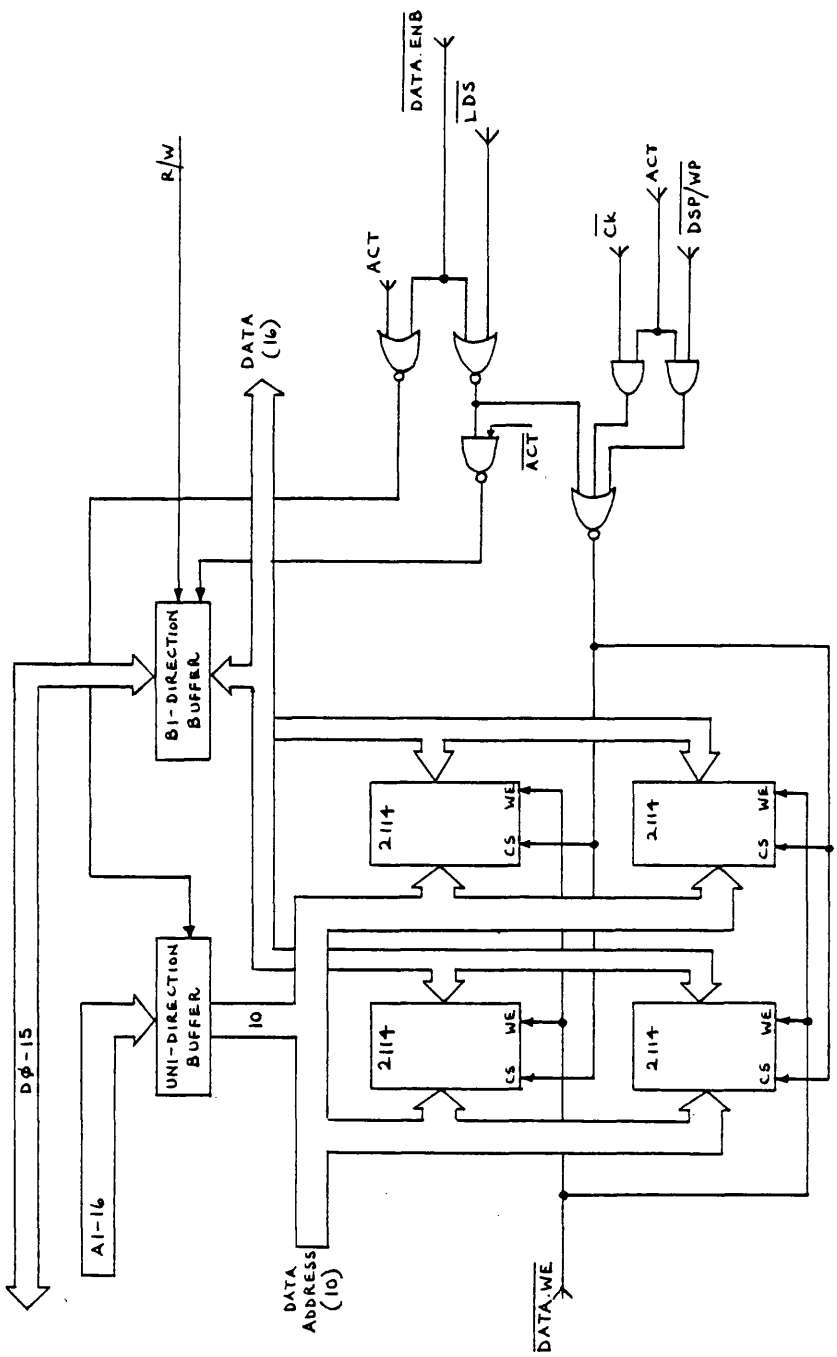


FIGURE III.6. DATA RAM AND LOGIC.

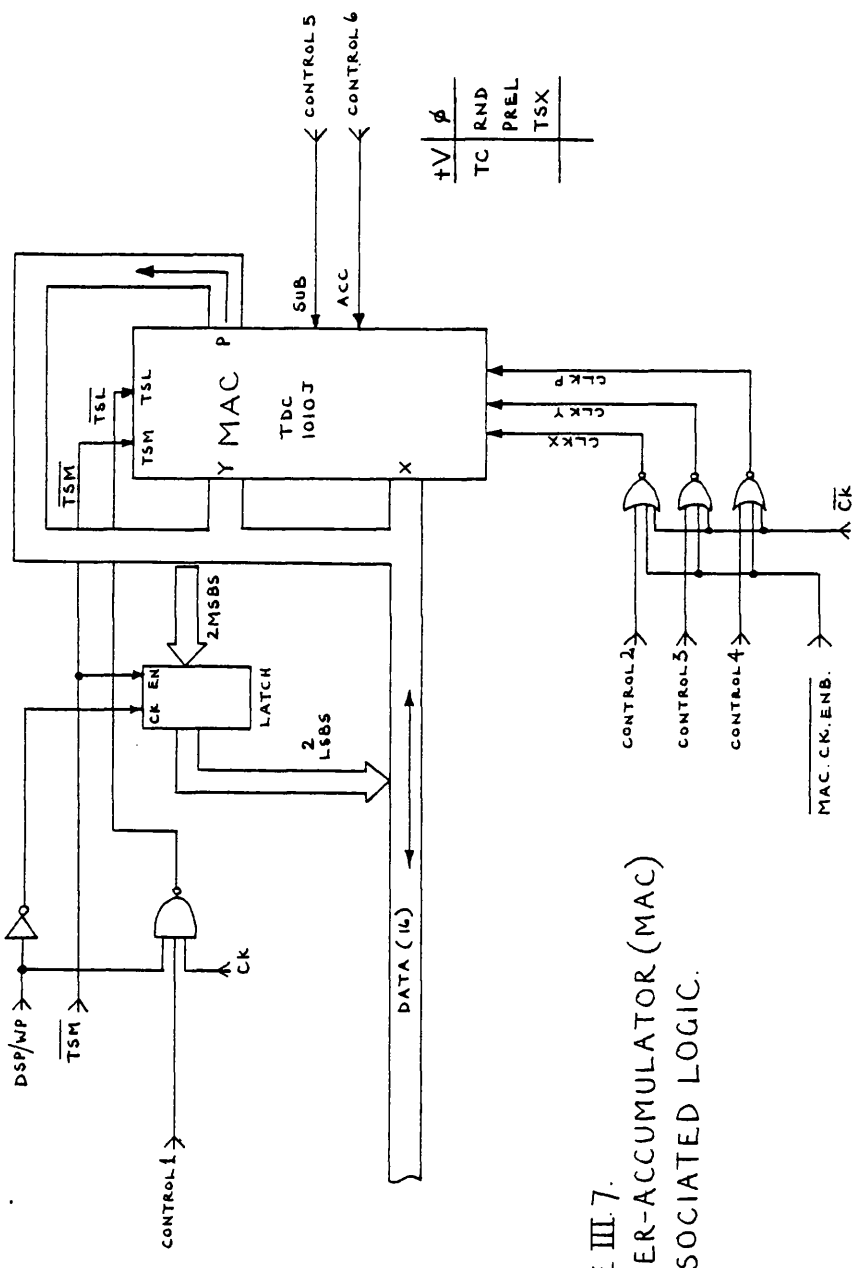


FIGURE III.7.
MULTIPLIER-ACCUMULATOR (MAC)
AND ASSOCIATED LOGIC.

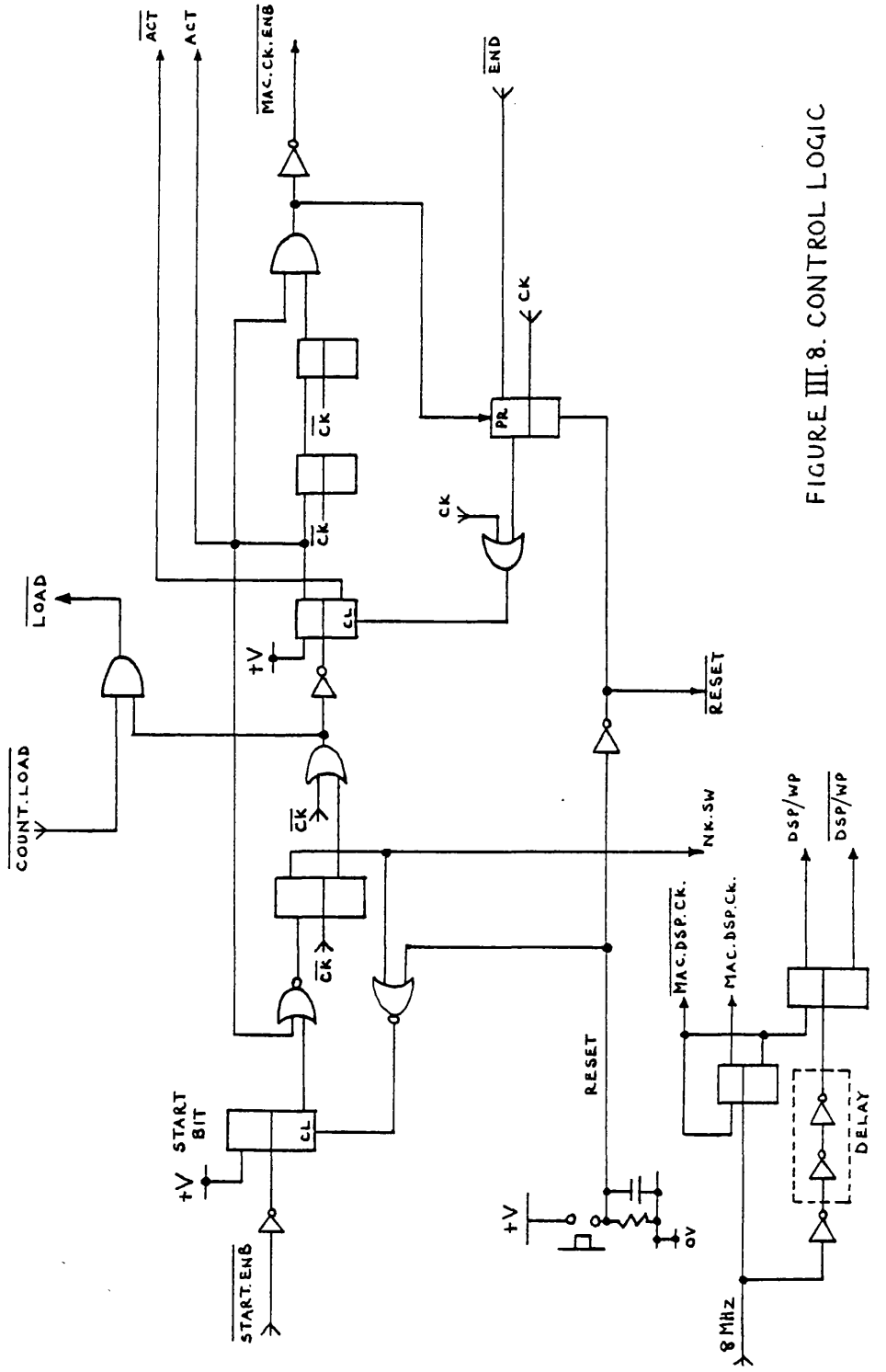


FIGURE III.8. CONTROL LOGIC

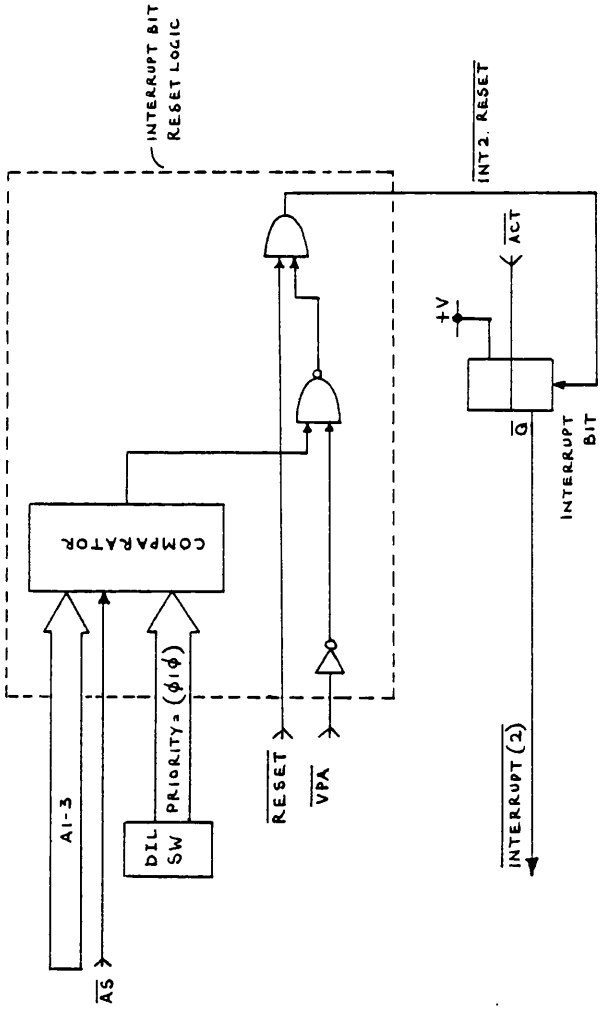


FIGURE III.9. INTERRUPT LOGIC

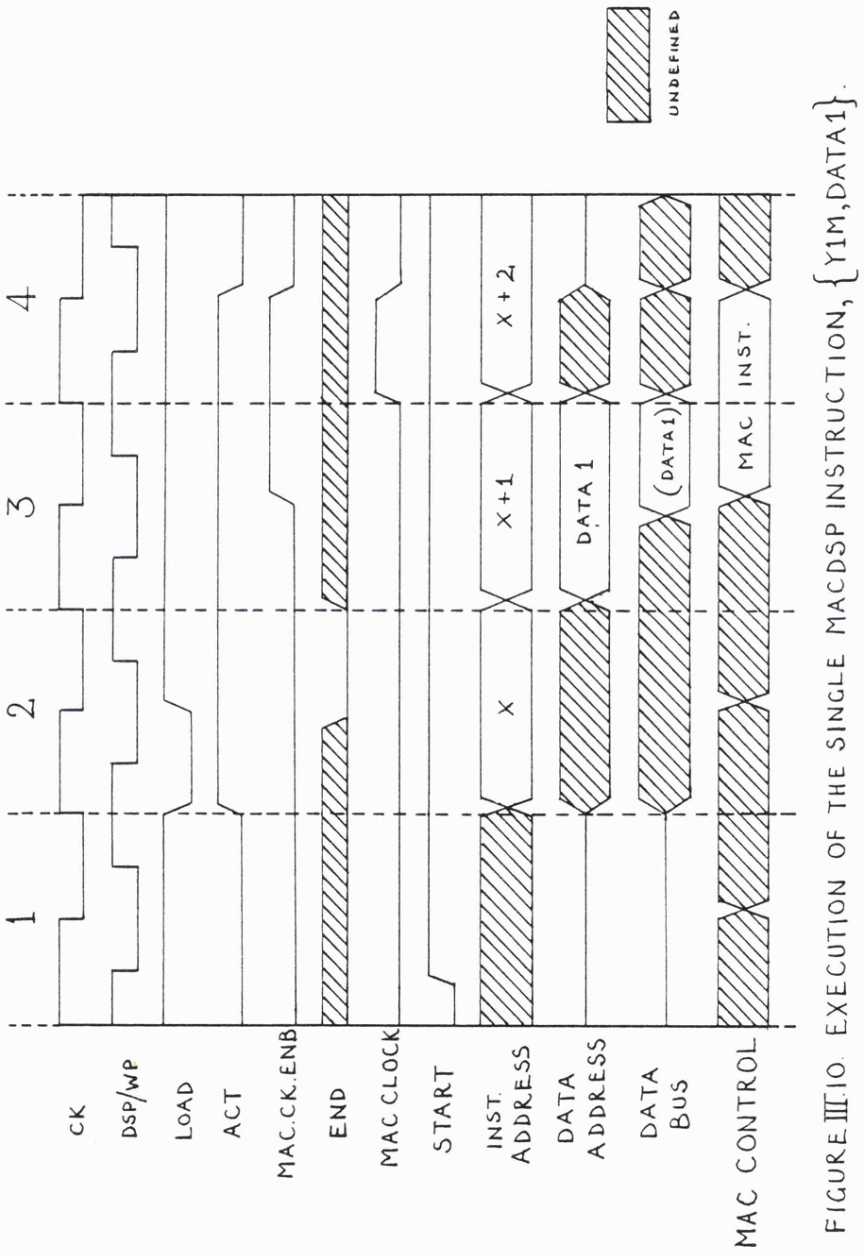


FIGURE III.10. EXECUTION OF THE SINGLE MACDSP INSTRUCTION, {Y1M,DATA1}.

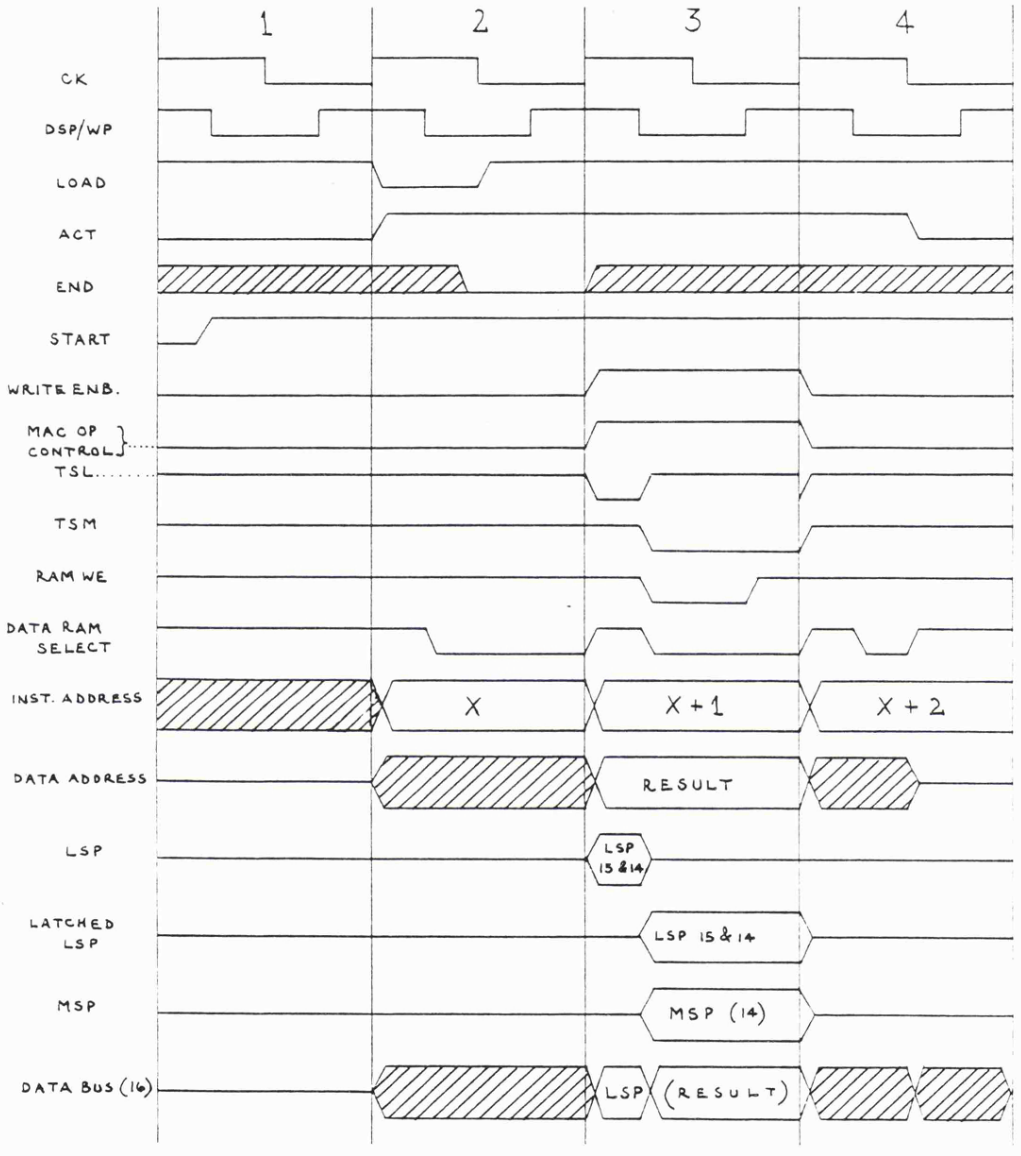


FIGURE III.10 CONTINUED. EXECUTION OF THE SINGLE MACDSP INSTRUCTION {PIN, RESULT}.

APPENDIX IV

ANALOG I/O LOGIC DESCRIPTION

Figure IV.1 shows the system diagram for the analog I/O circuit, and Figure IV.2 shows the timing diagram. The main components for the analog I/O circuit are:

- 12bit analog-to-digital convertor (AD574A).
- 12bit digital-to-analog convertor (AD567).
- Sample and Hold (AD582).

The overall circuit is split into 3 functional blocks which are described by the following:-

IV.1 MC68000 Bus Interface Logic (Figure IV.3)

The RESET- signal generated by the MACDSP, is used to reset the logic which enables the CK divider. Therefore deactivating the sampling process.

The MC68000 address lines are decoded to select the I/O board and to provide the following signals:

IO.START- Initiates the I/O process.

IO.STOP- Terminates the I/O process.

DA.SEL- Used to select the D/A.

AD.SEL- Used to select the A/D.

As with the MACDSP, an INTERRUPT BIT and associated logic is used on the I/O board, where:-

(1) The priority of the I/O board is 3.

(2) The INTERRUPT BIT activates the INTERRUPT(3) signal.

(3) The INTERRUPT BIT is set by the STS signal, which is generated by the A/D at the end of an A/D conversion.

IV.2 A/D Conversion Logic (Figure IV.4)

The A/D is configured for a 12bit bipolar conversion and the S/H is designed for a gain of +1.0. This allows for an input analog signal in the range $-10V \leq .. \leq +10V$.

The IO.DIV.ENB signal enables the CK divider, which is hardwired to generate the CONVERT- signal, a 12.5usec pulse produced every 125usec. This gives a fixed sampling frequency of 8kHz. The falling edge of CONVERT- is used to start the A/D conversion, during which the STS- signal is active. The STS signal has 2 functions:-

(1) To put the S/H into HOLD mode.

(2) Initiate I/O INTERRUPT processing.

When in HOLD mode the S/H switch is open, so that a previously acquired value is held constant during the A/D conversion. When the STS- signal is inactive the S/H is put into SAMPLE mode, so that the S/H switch is closed, and the S/H output follows the analog input signal.

When A/D conversion is complete, INTERRUPT(3) is activated, and the 12bit value is held in the A/D until the MC68000 reads it out. This value can only be held for an absolute maximum of 91usec, corresponding to the maximum acquisition time (the period in which the S/H acquires a new value).

IV.3 D/A Logic (Figure IV.5)

The D/A circuit is configured for a -5V<..<+5V bipolar output, and uses a 25pF capacitor to compensate for D/A output capacitance.

The D/A uses a double-buffered latch structure, so that the MC68000 can write a new value into the first latch without corrupting the value in the second latch. The second latch holds the value being converted.

The CONVERT- signal is used to transfer a value from the first latch to the second latch. Because the D/A process is continuous, the converted value will be on the D/A output a time period corresponding to the SETTLING TIME, after the falling edge of CONVERT-.

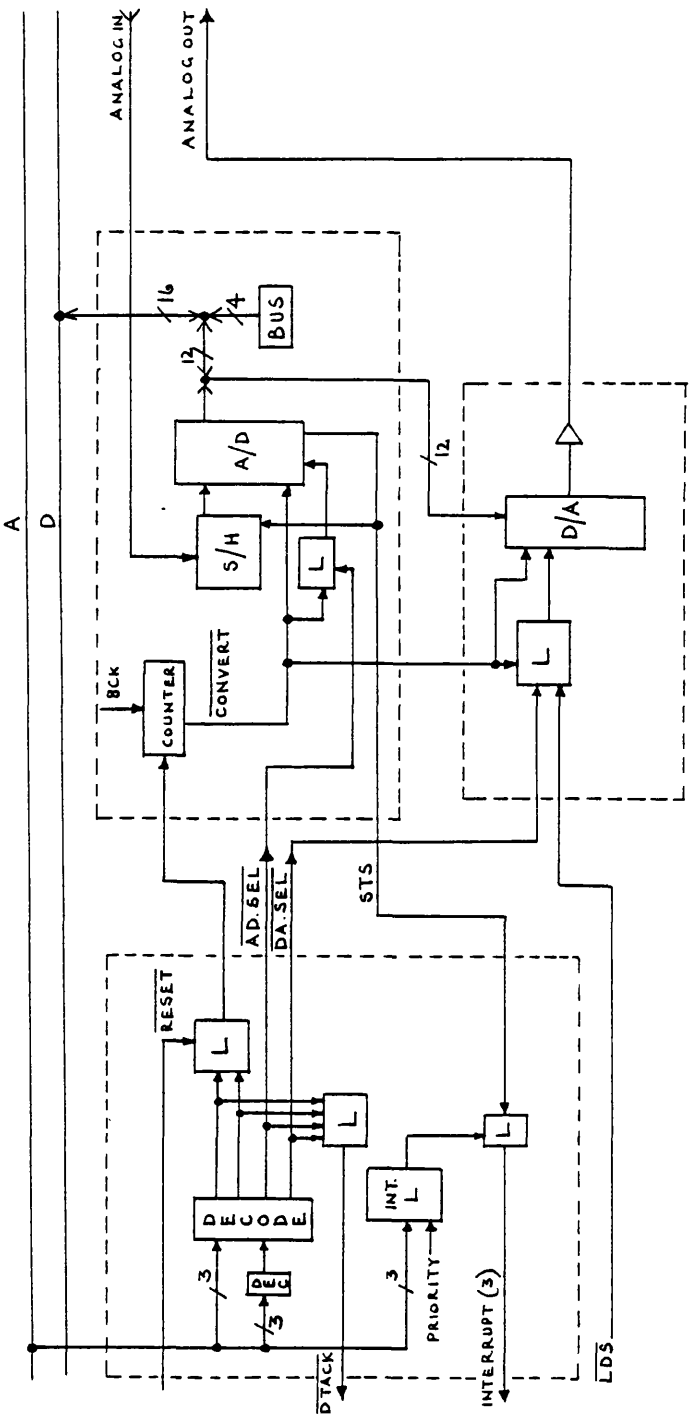


FIGURE IV.1. ANALOG I/O SYSTEM DIAGRAM.

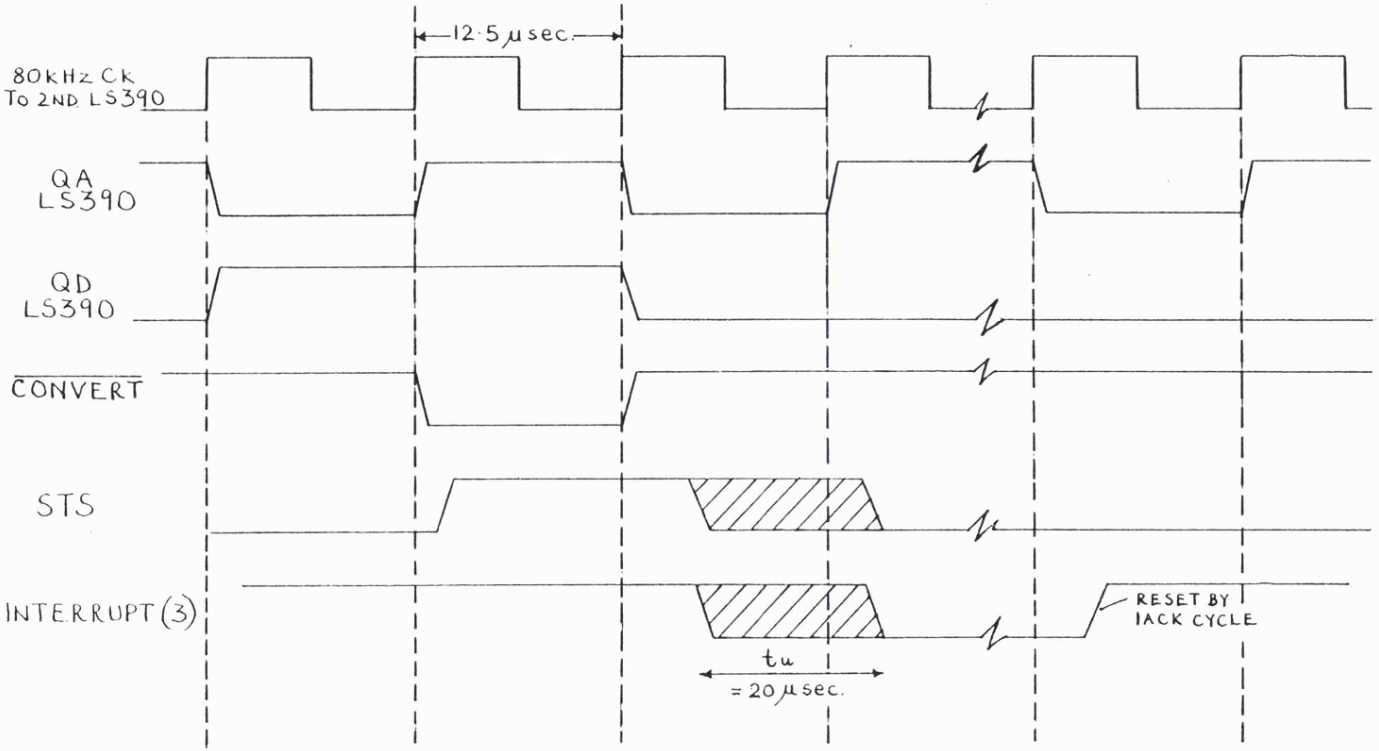


FIGURE IV. 2. ANALOG I/O TIMING DIAGRAM.

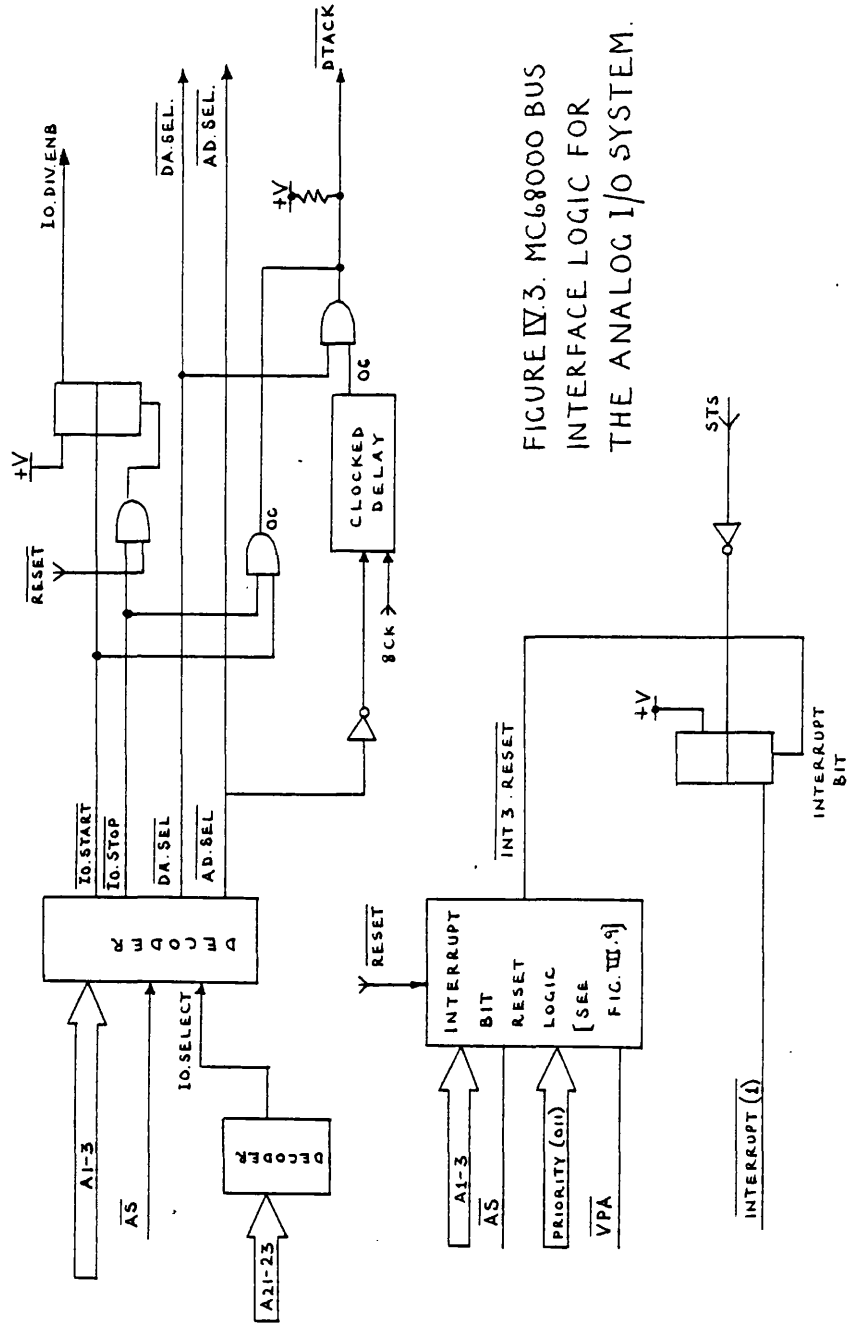


FIGURE IV.3. MC68000 BUS INTERFACE LOGIC FOR THE ANALOG I/O SYSTEM.

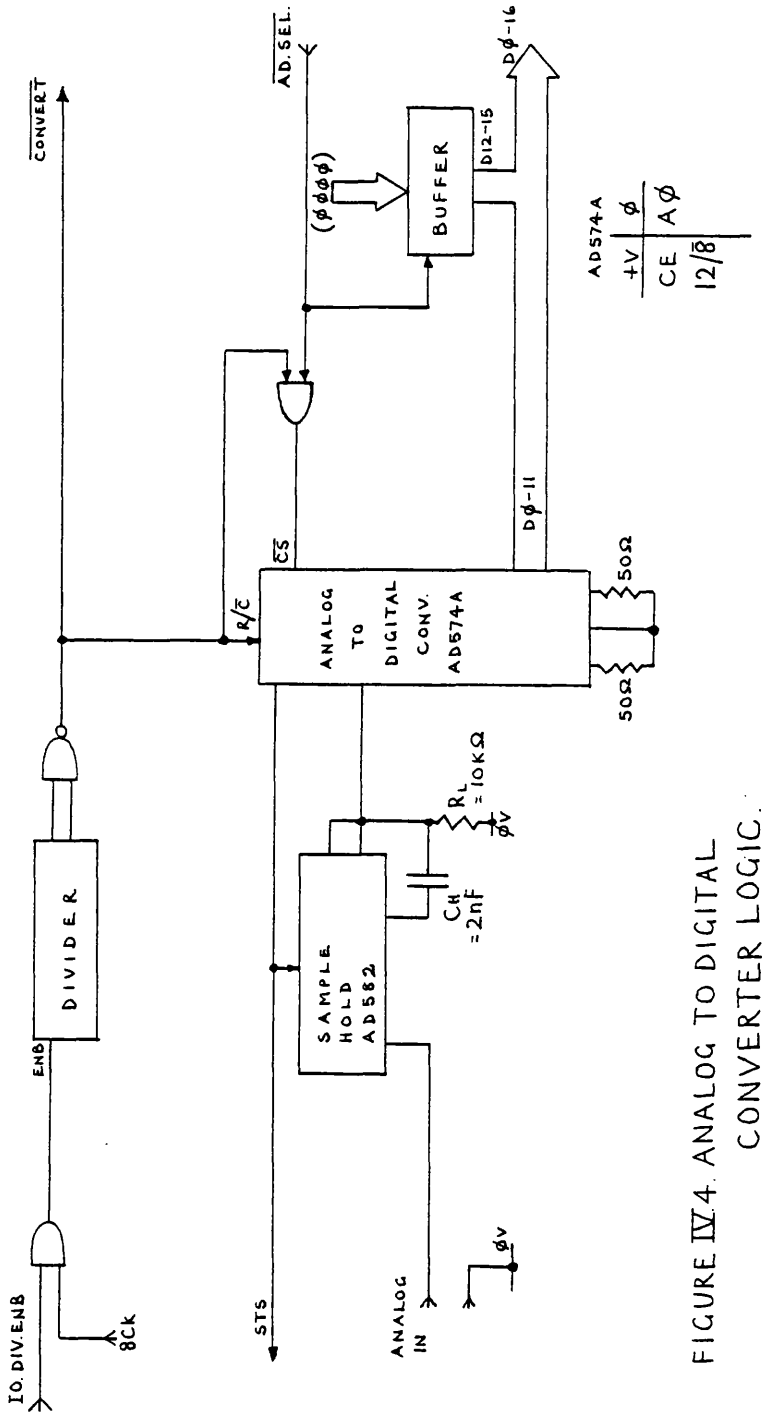


FIGURE IV.4. ANALOG TO DIGITAL CONVERTER LOGIC.

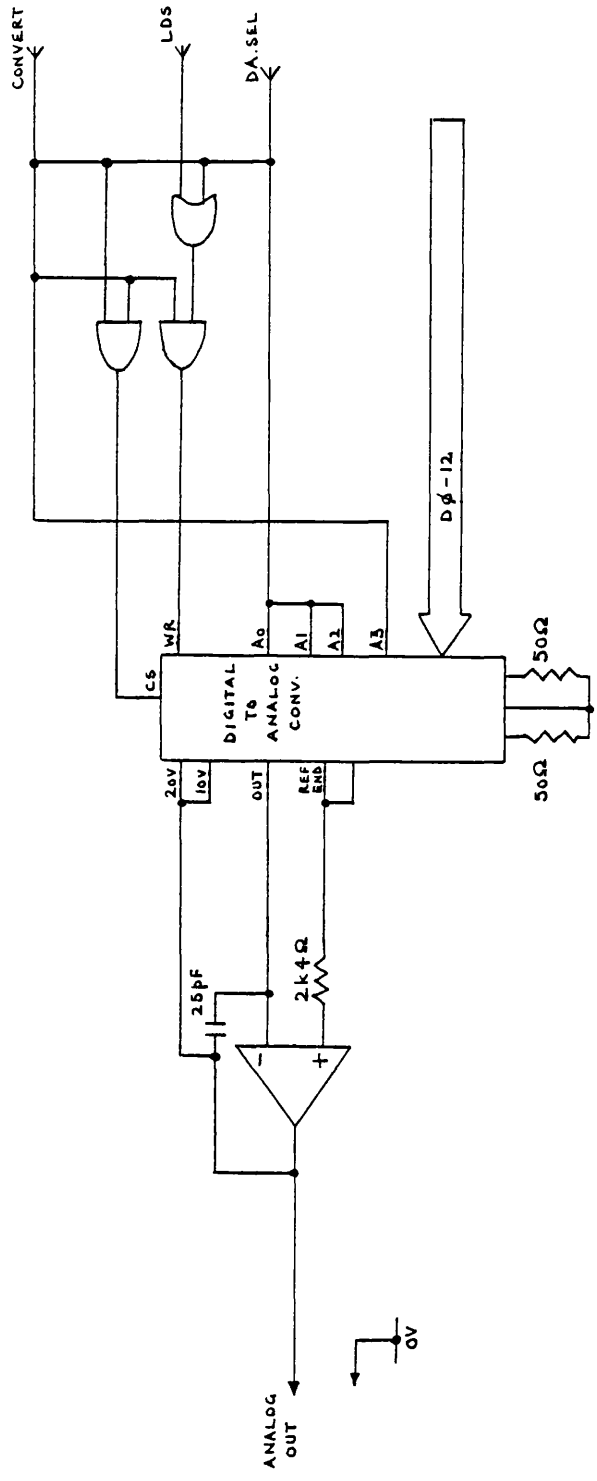


FIGURE IV.5. DIGITAL TO ANALOG CONVERTER LOGIC.

APPENDIX V

MACDSP ASSEMBLY' LANGUAGE AND ASSEMBLER

The MACDSP assembler has 2 main functions:-

(1) To detect syntax errors.

(2) To generate MACDSP code for disc storage or for direct loading into the MACDSP.

The syntax for the MACDSP assembly language is shown in Figure 3.3. The assembler works on a line-by-line basis, such that each line must be recognised and shown to be correctly placed in the program text, before proceeding. This simplified approach minimised the assembler complexity.

The entire assembler package had to be decomposed into separate parts called UNITs (see UCSD p-system manuals), where each UNIT is separately compiled and put into a LIBRARY.

The function of the various assembler parts are (see Figure V.1):-

MAIN1-Main calling program to which all higher-level commands are inputted. THis program cals up PROCC LINE, CODE LOAD, CODE ON DISC and DO ACTION. There are no ERROR recovery routines, so that an error terminates an assembly with an error message. This error message indicates the line number where the error has occurred.

PROC LINE-This UNIT controls the processing of each line of text. It ensures that each line recognised is correctly placed in the program text, as well as co-ordinating the acivities of PROCC CHAR and DO ACTION.

PROCC CHAR-This UNIT is passed the current character being processed as well as the current STATE. Using the CATEGORIES ARRAY (array of each allowable character) and the current STATE a 'state array' is indexed to produce the next STATE and any ACTION CODE. The state array is local to this UNIT only.

All unused locations in the state array contain -1, so that if this value is returned then a syntax error is signalled. This approach uses memory space inefficiently (a linked list approach would be more efficient) but the small state array size (12 by 15) meant this was unimportant.

DO ACTION-This procedure does a large case analysis using the ACTION CODE passed to it. Examples of some of the actions are:-

(1) Setting of LINE RECOG code indicating that a line has valid syntax as well as specifying the line type.

(2) Determination of MACDSP instruction values.

(3) Conversion of decimal numbers of addresses into binary equivalents.

DO ACTION maintains a code array in which the results of all actions are stored.

CODE ON DISC-Stores the code array on disc for later processing by the MACDSP loader.

CODE LOAD-Procedure for converting the code array into actual code for loading into the MACDSP. The setting of the END bits depends on whether single-stepping is to be used or not.

The following informally describes the semantics of the MACDSP assembly language (refer to Figure 3.3 for syntax):

COMMENTS

Comments can take an entire line by making the 1st character ';'.

Comments can also follow other assembly language text on the same line using ';'. All blanks are ignored.

DATA VALUES

The beginning of the data values section is indicated by '.D'. All data values must be in the range -1< ..<+1 (but not +1 exactly). ALl addresses are in the range 0-1023.

EXAMPLES

100,0.553 ---LOAD 0.553 INTO DATA RAM LOC.100

999,-.12 ---LOAD -0.12 INTO DATA RAM LOC.999

0,+0.001 ---LOAD 0.001 INTO DATA RAM LOC.0

COEFFICIENT/CONSTANT VALUES

The beginning of this section is indicated by '.C'. All coeff/constant values must be in the range -2 <..< +2 (but not +2 exactly). All addresses are in the range 0-1023.

EXAMPLES

23,1.6 ---LOAD 1.6 INTO DATA RAM LOC.23

177,-1.3 ---LOAD -1.3 INTO DATAS RAM LOC.177

MACDSP PROGRAM

The beginning of the program section is indicated by '.P'. Which is immediately followed by the start address line.

EXAMPLE:

S100 ---START ADDRESS=100

1st char; X-LOAD X REGISTER

Y-LOAD Y REGISTER

P-OUTPUT ACC. CONTENTS TO DATA RAM

N-NO EFFECT

2nd char: 1,2,3,4-OFFSET REGISTER NUMBER.

3rd char: M-MULTIPLY

A-MULTIPLY AND ACCUMULATE

S-MULTIPLY AND SUBTRACT

D-NO EFFECT

Main address: Range 0-1023.

EXAMPLES

X1D,300 ---LOAD X REGISTER WITH A VALUE FROM LOC. [OFFSET REG.1+300]. NO ARITHMETIC.

Y3S,943 ---LOAD Y REGISTER WITH A VALUE FROM LOC. [OFFSET REG.3+943]. THEN MULTIPLY X*Y AND SUBTRACT THE ACCUMULATOR CONTENTS FROM THE RESULT.

END OF TEXT

This is indicated by '.E'.

A complete commented MACDSP assembly language program is given in Figure 3.5.

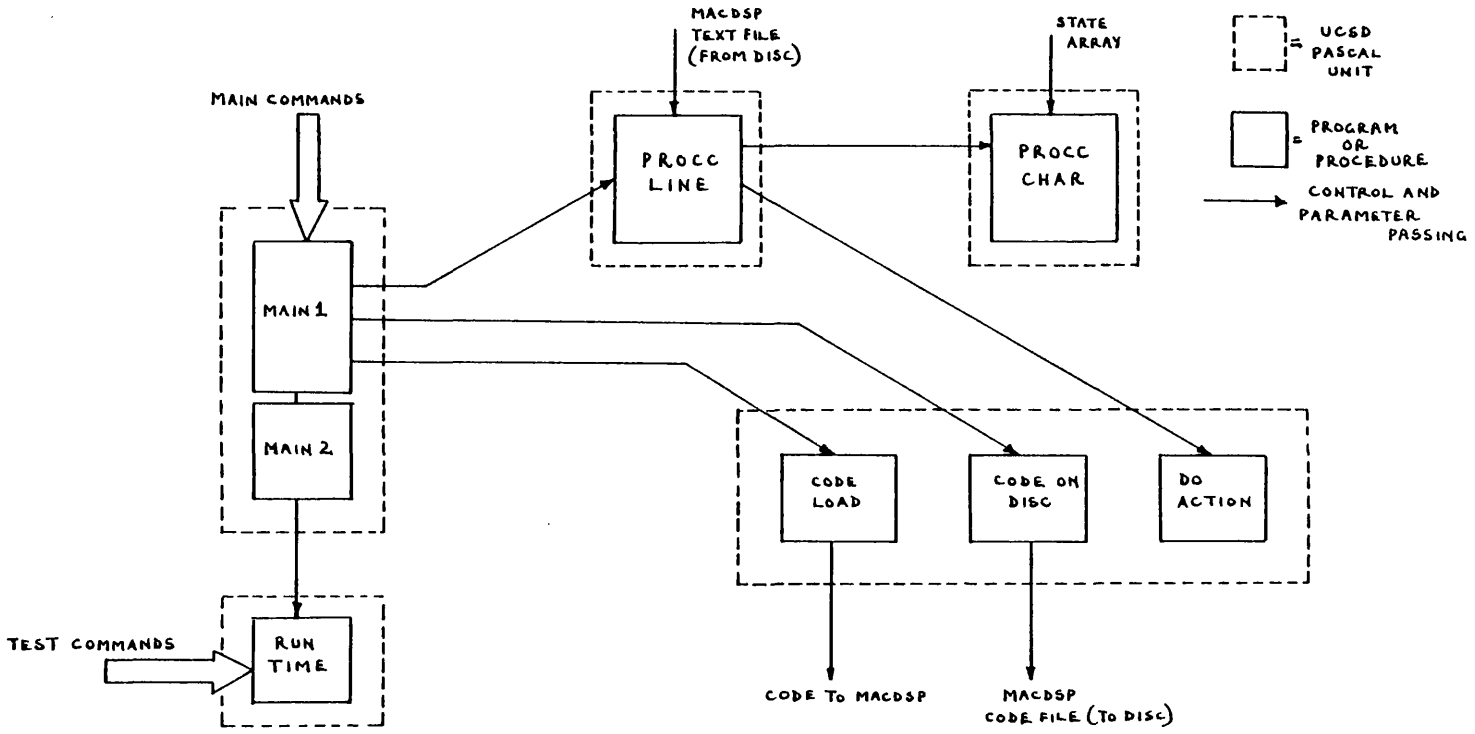


FIGURE V.1 SOFTWARE COMPONENTS FOR THE MACDSP ASSEMBLER AND DEBUGGER.

APPENDIX VI

MACDSP DEBUGGER

The debugger is made up to 2 separate parts MAIN2 and RUN TIME, which are integrated into the overall software environment (see Figure V.1). When a MACDSP program is to be debugged each MACDSP instruction is loaded with the END bit set to 0, so that the entire MACDSP program sequence can be single-stepped.

The UNIT RUN TIME is passed only 2 parameters from MAIN2, the MACDSP start address and the MACDSP end address. RUN TIME is then fed a list of parameters directly from the keyboard, these are used to set up the debugging. These parameters are as follows:-

TEST SIGNAL PARAMETERS

At present only a unit pulse input is implemented. The unit pulse parameters are:-

(1) Magnitude of unit pulse (range $0.0 \leq .. < +1.0$).

(2) Number of input samples (this sets the number of program sequence repeats).

(3) Input location (DATA RAM location where the input sample is to be loaded).

BREAKPOINTS

These are not breakpoints as used in most microprocessor debuggers, as the MACDSP program is always single-stepped. These breakpoints simply tell RUN TIME at which MACDSP instruction to display values to the screen. Options are:-

SINGLE-STEP --display values on every instruction.
LAST ONLY -- display values on the last instruction only.
SPECIFY -- Input up to 20 breakpoint locations (all relative to the start address).

MONITOR LOCATIONS

Up to 20 monitor locations (MACDSP DATA RAM locations) can be specified. On a breakpoint the decimal equivalent of the values stored at these locations will be displayed.

The debugger can also display the value of the accumulator on a breakpoint. This is done by using a single MACDSP instruction of the form:

P1D,1000 - END bit set to 0.

This instruction is placed in a specific location in MACDSP program RAM. On each breakpoint this single instruction is executed, so that the accumulator contents are loaded into loc.1000 without affecting the internal state of the MAC. The contents of loc.1000 can then be displayed.

Other parameters displayed are INSTRUCTION NUMBER and SAMPLE NUMBER. The entire process can be repeated as often as required.

NOTE:

Both offset register banks must be loaded with identical values (usually 0) to simulate the real-time running of the program. This is because in normal operation only one bank of values would be used when executing a multi-instruction sequence, but that when single- stepping the banks switch over for every instruction.

APPENDIX VII

MAC68 LOADERS

The entire loader package for the MC68000 and MACDSP routines is integrated into one package BU ENV. BU ENV takes the following inputs (see Figure VII.1):

(1) Name and start address for each MACDSP code file to be loaded.

(2) Name, relative start address and number of bytes for each MC68000 code file to be loaded. The relative start address is used to calculate the autovector value.

EXAMPLE:

If a MACDSP interrupt routine is specified, then:-

autovector(2)=(base address)+(relative start address)

where

base address=Fixed value referring to some reserved areas of SAGE II RAM.

relative start address=Number of bytes at the beginning of a MC68000 code file before an executable instruction is reached.

The total number of bytes in a MC68000 code file is given when the corresponding MC68000 assembly file is processed. This is done using the UCSD p-system package COMPRESS (see UCSD p-system manuals).

Given the above parameters BU ENV can create the complete real-time environment ready for execution. Once the execution command is given, the p-system can only be recovered by re-booting.

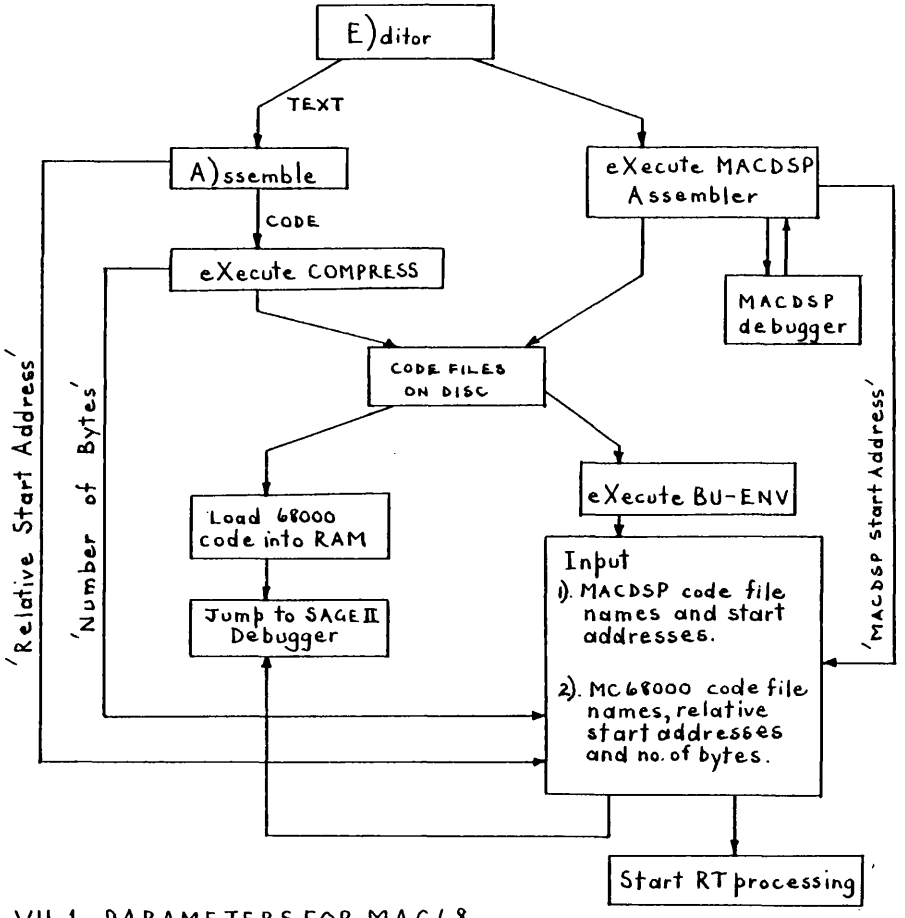


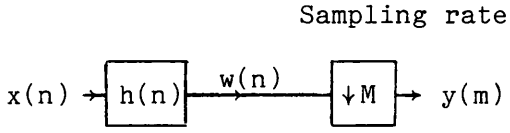
FIGURE VII.1. PARAMETERS FOR MAC68 CODE PRODUCTION.

APPENDIX VIII

COMPUTATION FOR 2-BAND FIR QUADRATURE MIRROR FILTERING

The following derives the equations and basic computational form for a 2-band Quadrature Mirror Filter (QMF) bank. The computational form can be used as a building block for more complicated sub-band schemes.

Consider firstly the basic equations for decimation and interpolation. For decimation, we have:-



where

$$w'(n) = \begin{cases} w(n) & n = 0, \pm M, \pm 2M, \ldots \\ \emptyset & \text{otherwise.} \end{cases} \qquad \text{(VIII.1)}$$

i.e. $w'(n) = w(n)$ at the sampling instants of $y(m)$, but is zero otherwise.

Therefore,

$$w'(n) = w(n) \left\{ \frac{1}{M} \sum_{\ell=0}^{M-1} e^{j2\pi \ell n/M} \right\} \quad -\infty<n<\infty \qquad \text{(VIII.2)}$$

We now write the z-transform of $y(m)$ as

$$Y(z) = \sum_{m=-\infty}^{\infty} y(m) z^{-m}$$

$$= \sum_{m=-\infty}^{\infty} W'(Mm) z^{-m} \qquad \text{(VIII.3)}$$

Using (VIII.2) we have,

$$Y(z) = \frac{1}{M} \sum_{\ell=0}^{M-1} W(e^{-j2\pi\ell/M} . z^{1/M}) \qquad \text{(VIII.4)}$$

Since,

$$W(z) = H(z)\ X\ (z) \qquad \text{(VIII.5)}$$

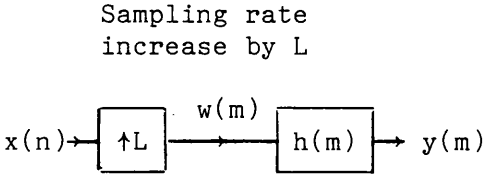
the equation for Y(z) becomes,

$$Y(z) = \frac{1}{M} \sum_{\ell=0}^{M-1} H(e^{-j2\pi\ell/M} . z^{1/M}) . X\ (e^{-j2\pi\ell/M} . z^{1/M}) \qquad \text{(VIII.6)}$$

For M=2, eqn. (VIII.6) becomes,

$$Y(z) = \tfrac{1}{2} [H(z^{\frac{1}{2}}).X(z^{\frac{1}{2}}) + H(z^{-\frac{1}{2}}).X\ (z^{-\frac{1}{2}})] \qquad \text{(VIII.7)}$$

Consider now the configuration for interpolation:-



where,

$$w(m) = \begin{cases} x(m/L), & m=0,\ \pm L,\ \pm 2L \\ 0\ , & \text{otherwise.} \end{cases} \qquad \text{(VIII.8)}$$

The resultant signal w(m) has the z-transform,

$$W(z) = \sum_{m=-\infty}^{\infty} w(m) z^{-m}$$

$$= \sum_{m=-\infty}^{\infty} x(m) z^{-mL}$$

$$= X(z^{L}) \qquad \text{(VIII.9)}$$

Therefore,

$$Y(z) = H(z).X(z^{L}) \qquad \text{(VIII.10)}$$

For L=2, eqn. (VIII.10) becomes

$$Y(z) = H(z).X(z^{2}) \qquad \text{(VIII.11)}$$

Having derived the basic equations for interpolation and decimation, we can consider a 2-band QMF bank.

A 2-channel system in which the input signal x(n) sampled at frequency fs, with fs = 1/T = Ws/2 π, is split into two half-band channels with cut-off frequency fs/4. The lower half-band channel is derived by low-pass filtering the input signal x(n) with a M tap FIR filter, H, with impulse response h(n) and z-transform H(z). The high-pass filter $h_1(n)$ is derived from h(n) by the simple low-pass to high-pass transformation:-

$$h_1(n) = (-1)^{n} h(n) \qquad \text{(VIII.12)}$$

[See Figure VIII.1].

Thus the z-transform of $h_1(n)$ is H(-z).
From eqn. (VIII.7) we can obtain the low-pass and high-pass transmit filter outputs, $Y_1(z)$ and $Y_2(z)$ respectively,

$$Y_1(z) = \tfrac{1}{2}\ [H(z^{\frac{1}{2}}).\ X(z^{\frac{1}{2}}) + H(-z^{\frac{1}{2}}).\ X(-z^{\frac{1}{2}})] \qquad \text{(VIII.13)}$$

$$Y_2(z) = \tfrac{1}{2}\ [H(-z^{\frac{1}{2}}).\ X(z) + H(z^{\frac{1}{2}}).\ X(-z^{\frac{1}{2}})] \qquad \text{(VIII.14)}$$

Using eqn. (VIII.11), we obtain the output signal,

$$\tilde{X}(z) = Y_1(z^2).H(z) - Y_2(z^2).H(-z) \qquad \text{(VIII.15)}$$

which, using eqns. (VIII.13) and (VIII.14) becomes,

$$\tilde{X}(z) = \tfrac{1}{2} X(z).[H^2(z) - H^2(-z)] \qquad \text{(VIII.16)}$$

Consequently for perfect reconstruction of x(n),

$$\tfrac{1}{2}[H^2(z) - H^2(-z)] = 1, \text{ for all } z \qquad \text{(VIII.17)}$$

Evaluating eqn. (VIII.16) on the unit circle gives,

$$\tilde{X}(e^{jwT}) = \tfrac{1}{2} X(e^{jwT}).[H^2(e^{jwT}) - H^2(e^{j(w+(ws/2))T})] \qquad \text{(VIII.18)}$$

Choosing H as a symmetrical FIR filter, its Fourier transform $H(e^{jwT})$ can be expressed as:-

$$H(e^{jwT}) = |H(e^{jwT})|e^{-j(M-1)\pi w/ws} \qquad \text{(VIII.19)}$$

Substituting eqn. (VIII.19) into (VIII.18), and letting $H(w) = |H(e^{jwT})|$ gives,

$$\tilde{X}(e^{jwT}) = \tfrac{1}{2} X(e^{jwT}).e^{-j2\pi(M-1)w/ws} .[H^2(w) - H^2(w+\tfrac{ws}{2})e^{-j(M-1)\pi}] \qquad \text{(VIII.20)}$$

As x(n) cannot be perfectly reconstructed when M is odd, M must be even. Therefore,

$$\tilde{X}(e^{jwT}) = \tfrac{1}{2} X(e^{jwT}).e^{-j2\pi(M-1)w/ws}[H^2(w) + H^2(w+\frac{ws}{2})] \qquad \text{(VIII.21)}$$

so that,

$$H^2(w) + H^2(w+\frac{ws}{2}) = 1 \qquad \text{(VIII.22)}$$

is a necessary condition for perfect reconstruction of x(n).

To summarise, given the configuration of Figure VIII.1, h(n) must have the properties:-

(a) Symmetrical even order FIR filter.

(b) Frequency response given by eqn. (VIII.22).

The configuration shown in Figure VIII.1 can be implemented directly using FIR filters with the above properties. But, by using polyphase representations for 2-band decimation and interpolation, the same filters can be transformed to give equivalent structures using fewer computations. Relevant polyphase structures are given in Figure VIII.2, where [C-8],

$$p0(n) = h(2n) = p0'(n) \qquad \text{(VIII.23)}$$

$$p1(n) = h(2n+1) = - p1'(n) \qquad \text{(VIII.24)}$$

From these representations the structure given in Figure VIII.3 can be derived. This structure has approximately half the computational load of the unmodified structure.

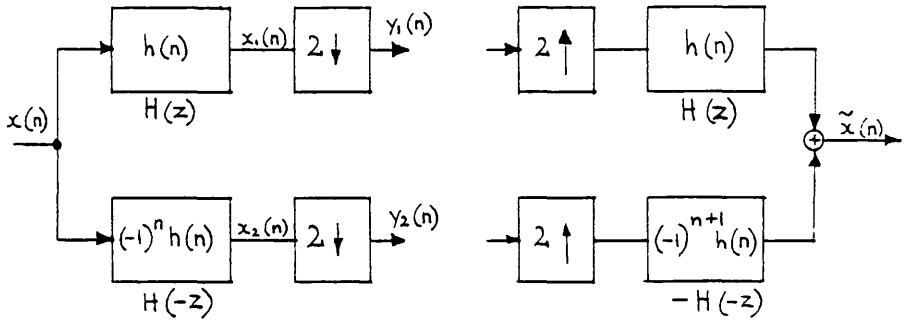


FIGURE VIII. 1. Two-band FIR QMF

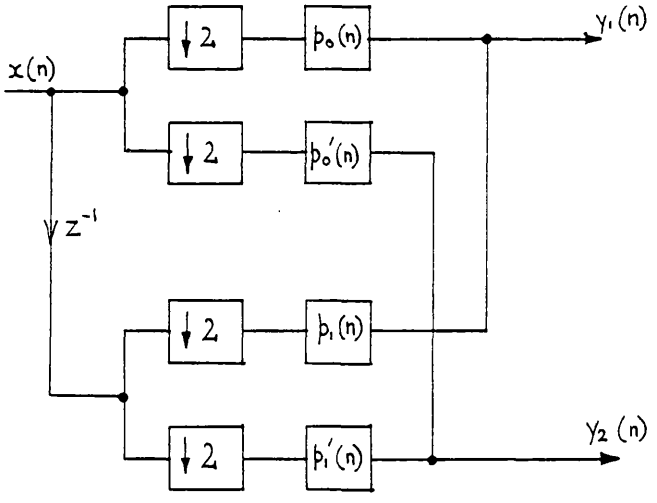


FIGURE VIII. 2. Polyphase equivalent for the Transmit side of a 2-band FIR QMF.

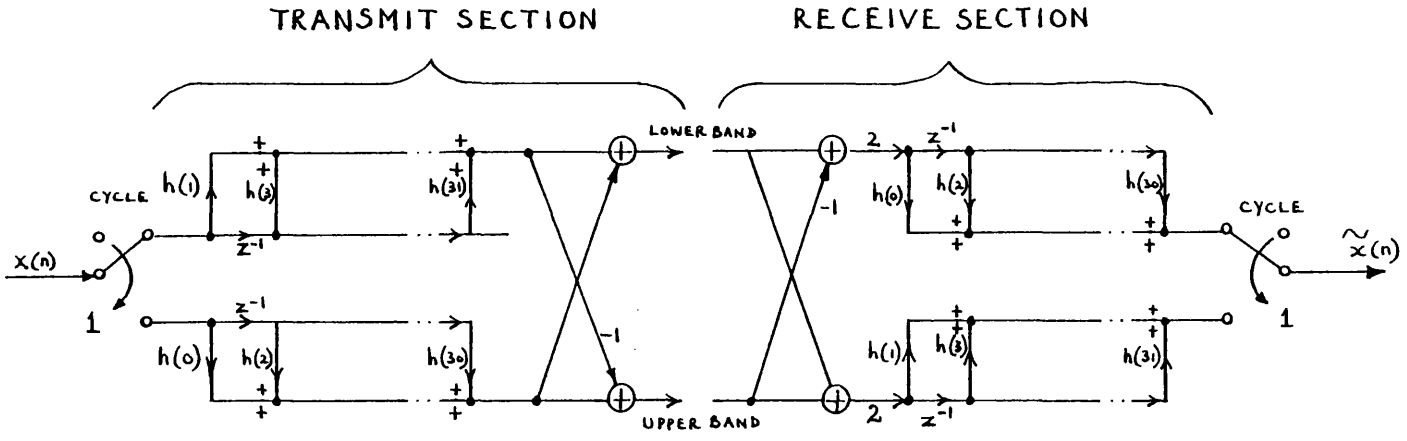


FIGURE VIII.3. COMPUTATIONAL STRUCTURES FOR 2-BAND FIR QUADRATURE MIRROR FILTERS

REFERENCES

A-1 J. Allen.
Computer Architecture for Signal Processing.
Proc. IEEE, vol.63, pp.624-633, Apr. 1975.

A-2 P. Alexander.
Array Processor Design Concepts.
Computer Design, Dec. 1981, pp.163-172.

A-3 G.R. Andrews, F.B. Schneider.
Concepts and Notations for Concurrent Programming.
Computing Surveys, vol.15, no.1, pp.3-43, Mar. 1983.

A-4 A.V. Aho, J.D. Ullman.
Principles of Compiler Design.
Addison-Wesley, 1977.

B-1 S. Bal, A Kaminker, Y. Lavi, A.M. Menachem, Z. Soha.
The NS16000 Family - Advances in Architecture and Hardware.
Computer, June 1982, pp.58-67.

B-2 R. Bernhard.
More Hardware means less Software.
IEEE Spectrum, 1981.

B-3 R.W. Blasco.
V-MOS Chip Joins Microprocessor to Handle Signals in Real-Time.
Electronics, Aug.30 1979, pp.131-138.

B-4 R.G. Barr, J.A. Becker, W.P. Lidinsky, V.V. Tantillo.
A Research-Orientated Dynamic Microprocessor.
IEEE Trans. Comput., vol.C-22, pp.976-985, Nov.1973.

B-5 J.R. Boddie et al.
Digital Signal Processor: Architecture and Performance.
Bell Syst. Tech. J., vol.60, no.7, pp.1449-1462, Sept.1981.

B-6 J.R. Boddie et al.
Digital Signal Processor: Adaptive Pulse-Code-Modulation Coding.
Bell Syst. Tech. J., vol.60, no.7, pp.1547-1561, Sept.1981.

B-7 J. Backus.
Can Programming Be Liberated from the von Neumann Style?
A FUnctional Style and Its Algebra of Programs.
Communications ACM, vol.21, no.8, pp.613-641, Aug.1978.

B-8 I. Barron, P. Cavill, D. May, P. Wilson.
The Transputer.
Electronics, Nov.17 1983, pp.109-115.

B-9 J.L. Baer.
Computer Systems Architecture.
Addison-Wesley.

B-10 D.P. Siewiorek, C.G. Bell, A. Newell.
Computer Structures: Principles and Examples.
McGraw-Hill, Inc. 1982.

B-11 P. Bylanski, T.W. Chang.
Advances in Speech Coding for Communications.
GEC Journal of Research, Vol 2, no.1, 1984 pp.16-22.

C-1 R.H. Cushman.
Digital Signal Processing Advances Slowly But Steadily.
EDN, pp.60-72, July 7, 1983.

C-2 R.E. Crochiere, A.V. Oppenheim.
Analysis of Linear Digital Networks.
Proc. IEEE, vol.63, pp.581-595, Apr. 1975.

C-3 H.G. Cragon.
The Elements of Single-Chip Microcomputer Architecture.
Computer, Oct.1980, pp.27-41.

C-4 R.H. Cushman.
Signal Processing Design Awaits Digital Takeover.
EDN, June 24, 1981, pp.119-128.

C-5 R.S. Cheung, R.L. Winslow.
High Quality 16kb/s Voice Transmission:
The Subband Coder Approch.
Proc. IEEE Int. Conf. ASSP 1980, pp.319-322.

C-6 R.E. Crochiere.
On the Design of Sub-band Coders for Low-Bit-Rate Speech Communication.
Bell Syst. Tech. J., vol.56, no.5, pp.747-770, May-June 1977.

C-7 R.E. Crochiere, M. Randolph, J.W. Upton, J.D. Johnston.
Real-Time Implementation of Sub-Band Coding on a Programmable Integrated Circuit.
Proc. IEEE Int. Conf. ASSP 1981, pp.455-458.

C-8 R.E. Crochiere, L.R. Rabiner.
Interpolation and Decimation of Digital Signals - A Tutorial Review.
Proc. IEEE, vol.69, no.3, pp.300-331, Mar. 1981.

C-9 R.E. Crochiere, R.V. Cox, J.D. Johnston.
Real-Time Speech Coding.
IEEE Trans. Commun., vol.COM-30, no.4, pp.621-634, April 1982.

C-10 R.E. Crochiere.
Digital Signal Processor, Sub-Band Coding.
Bell Syst. Tech. J., vol.60, no.7, pp.1633-1653, Sept.1981.

C-11 Y.M. Chong.
Data Flow Chip Optimises Image Processing.
Computer Design, Oct.15, 1984, pp.97-103.

D-1 A.C. Davies.
Trade-offs in Fixed-Point Multiplication Algorithms for Microprocessors.
IEE Comp. & Digital Tec., June 1979, vol.2, no.3, pp-105-112.

D-2 K. Davies, F. Ris.

Real-Time Signal Processor Software Support.

IBM J. Res. Develop., vol.26, no.4, pp.432-439, July 1982.

D-3 A.L. Davis, R.M. Keller.

Data Flow Program Graphs.

Computer, Feb. 1982, pp.26-41.

E-1 P.M. Ebert, J.E. Mazo, M.G. Taylor.

Overflow Oscillations in Digital Filters.

Bell System Tech J., vol.48, pp.2999-3020, 1969.

E-2 D. Esteban, C. Galand.

Application of Quadrature Mirror Filters to Split Band Voice Coding Schemes.

Proc. IEEE Int. Conf. ASSP 1977, pp.191-195.

F-1 M.J. Flynn.

Directions and Issues in Architecture and Language.

Computer, Oct. 1980, pp.5-22.

F-2 S.L. Freeny.

Special-Purpose Hardware for Digital Filtering.

Proc. IEEE, vol.63, pp.633-648, Apr. 1975.

F-3 J.R. Fisher.

Architecture and Applications of the SPS-41 and SPS-81 Programmable Digital Signal Processors.

EASCON 74 Rec., Oct.7-9, 1974, pp.674-678.

G-1 A. Gupta, H.D. Toong.
Microprocessors - The First Twelve Years.
Proc. IEEE, vol.71, no.11, pp.1236-1256, Nov. 1983.

G-2 A. Gupta, H.D. Toong.
An Architectural Comparison of 32-bit Microprocessors.
IEEE Micro, Feb. 1983, pp.9-22.

G-3 B. Gold, I.L. Lebow, P.G. McHugh, C.M. Rader.
The FDP, a Fast Programmable Signal Processor.
IEEE Trans. Comput., vol.C-20, pp.33-38, Jan. 1971.

G-4 D. Geist.
Enhance M6800 Power with Fast Multiply Hardware.
Electronic Engineering, Sept. 1978, pp.64-65.

G-5 D.J. Goodman, R.M. Wilkinson.
A Robust Adaptive Quantizer.
IEEE Trans. Commun., Nov. 1975, pp.1362-1365.

G-6 C.R. Galand, H.J. Nussbaumer.
New Quadrature Mirror Filter Structures.
IEEE Trans. ASSP-32, no.3, pp.522-531, June 1984.

G-7 H. Gethoffer.
SIPROL: A High Level Language for Digital Signal Processing.
Proc. IEEE Int. Conf. ASSP 1980, pp.1056-1059.

G-8 D. Garde.
Single-port Multiplier Reduces DSP-System Costs.
EDN, Jan.12 1984, pp.277-286.

H-1 J.V. Harshman.
Architecture of a Programmable Digital Signal Processor.
NaT. Telecommun. Conf. Rec., Dec.2-4, pp.496-500, 1974.

H-2 R. Hausman, P. Cannon.
Array processor Achieves 100 MFLOPS.
Computer Design, Feb. 1984, pp.105-113.

H-3 J.N. Holmes.
A Survey of Methods for Digitally Encoding Speech Signals.
The Radio and Electronic Engineer, vol.52, no.6, pp.267-276, June 1982.

J-1 N.S. Jayant.
Digital Coding of Speech Waveforms:
PCM, DPCM, and DM Quantizers.
Proc. IEEE, vol.62, no.5, pp.611-632, May 1974.

J-2 J.D. Johnston.
A Filter Family Designed for Use in Quadrature Mirror Filter Banks.
Proc. IEEE Int. Conf. ASSP, April 1980, pp.291-294.

J-3 K. Jensen, N. Wirth.
PASCAL User Manual and Report.
Springer-Verlag, 1974.

K-1 G.E. Kopec.
The Signal Representation Language SRL.
Proc. IEEE Int. Conf. ASSP 1983, pp.1168-1171.

K-2 S. Kung.
On Supercomputing with Systolic/Wavefront Array Processors.
Proc. IEEE, vol.72, no.7, pp.867-894, July 1984.

L-1 B. Liu.
Effect of Finite Word Length on the Accuracy of Digital Filters - A Review.
IEEE Trans. Circuit Theory, vol. CT-18, pp.670-677, Nov.1971.

M-1 S.S. Magar.
Signal Processing Chips Invite Design Comparisons.
Computer Design, Apr. 1984, pp.179-184.

M-2 F. Mintzer, A Peled.
A Microprocessor for Signal Processing, the RSP.
IBM J. Res. Develop., vol.26, no.4, pp.413-423, July 1982.

M-3 L.R. Morris.
Automatic Generation of Time Efficient Digital Signal processing Software.
IEEE Trans. ASSP-25, no.1, pp.74-79, Feb. 1977.

M-4 F. Mintzer.
Parallel and Cascade Microprocessor Implementations for Digital Signal Processing.
IEEE Trans. ASSP-29, no.5, pp.1018-1027, Oct. 1981.

M-5 L.R. Morris.
A Tale of Two Architectures:
TI TMS320 SPC vs. DEC Micro/J-11.
Int. Conf. ASSP 1983, pp.1200-1203.

N-1 H.T. Nagle, Jr., V.P. Nelson.
Digital Filter Implementation on 16-bit Microcomputers.
IEEE Micro, Feb. 1981, pp.23-41.

O-1 A.V. Oppenheim, R.W. Schafer.
Digital Signal Processing.
Prentice-Hall Inc., 1975.

P-1 D.A. Patterson, C.H. Sequin.
A VLSI RISC.
Computer, Sept. 1982, pp.8-21.

R-1 C.V. Ramamoorthy, H.F. Liu.
Pipeline Architecture.
Computing Surveys, vol.9, no.1, pp.61-102, March 1977.

S-1 A. Sawai.
Programmable LSI Digital Signal Processor Development.
VLSI Systems and Computations, Computer Science Press 1981.
pp.29-40.

S-2 M. Stauffer.
Fast Fourier Transforms using Arithmetic Processors.
Electronic Product Design, Jan. 1981, pp.36-41.

T-1 H.D. Toong, A. Gupta.
An Architectural Comparison of Contemporary 16-bit Microprocessors.
IEEE Micro, May 1981, pp.26-37.

T-2 J.S. Thompson, S.K. Tewksbury.
LSI Signal Processor Architecture for Telecommunications Applications.
IEEE Trans. ASSP-30, no.4, pp.613-631, Aug. 1982.

T-3 P. Thirion.
Digital Signal Processing with Program Synchronisation Between Two Microprocessors.
IEEE Trans. Commun., vol. COM-26, no.5, pp.513-517, May 1978.

T-4 R.D. Tennent.
Principles of Programming Languages.
Prentice Hall International Inc., 1981.

W-1 P.D. Welch.
A Fixed-Point Fast Fourier Transform Error Analysis.
IEEE Trans. Audio Electroac., vol. AU-17, pp.151-157, June 1969.

W-2 A.J. Weissburger.
Analysis of Multiple-Microprocessor System Architectures.
Computer Design, June 1977, pp.151-163.

W-3 W.A. Wulf.
Compilers and Computer Architecture.
Computer, July 1981, pp.41-47.

W-4 I. Watson, J. Gurd.
A Practical Data Flow Computer.
Computer, Feb. 1982, pp.51-57.

Z-1 M.P.Zoccoli, A.C. Sanderson.
The RAPID Bus Multiprocessor System.
Computer Design, Nov. 1981, pp.189-200.